Constraints, , 1?? ()

Improving linear constraint propagation by changing constraint representation

WARWICK HARVEY* wharvey@csse.monash.edu.au
School of Computer Science and Software Engineering, Monash University, Clayton VIC 3168, Australia

PETER J. STUCKEY pjs@cs.mu.oz.au
Dept. of Computer Science and Software Engineering, University of Melbourne, Parkville VIC 3052, Australia

Abstract. Propagation based finite domain solvers provide a general mechanism for solving combinatorial problems. Different propagation methods can be used in conjunction by communicating through the domains of shared variables. The flexibility that this entails has been an important factor in the success of propagation based solving for solving hard combinatorial problems. In this paper we investigate how linear integer constraints should be represented in order that propagation can determine strong domain information. We identify two kinds of substitution which can improve propagation solvers, and can never weaken the domain information. This leads us to an alternate approach to propagation based solving where the form of constraints is modified by substitution as computation progresses. We compare and contrast a solver using substitution against an indexical based solver, the current method of choice for implementing propagation based constraint solvers, identifying the relative advantages and disadvantages of the two approaches. In doing so we investigate a number of choices in propagation solvers and their effects on a suite of benchmarks.

Keywords: Finite domain constraint solving, propagation methods, linear integer constraints

1. Introduction

Propagation based finite domain solvers provide a general mechanism for solving combinatorial problems. Different propagation methods can be used in conjunction by communicating through the domains of shared variables. The flexibility that this entails has been an important factor in the success of propagation based solving for solving hard combinatorial problems. The most primitive and common constraints that are used within propagation based solvers are linear integer constraints. Most simple constraints fall into this category and usually they form the bulk of constraints used in modelling a problem. More complicated propagation constraints are nonlinear constraints, usually restricted to a few forms like $X \times Y = Z$ and $X^2 = Z$. Modern propagation based constraint solvers like CHIP [1, 11] and ILOG SOLVER [10] also include libraries of specialized propagation methods for complex constraints such as `cumulative` and `cycle`. Because of the uniform communication mechanism complex multi-faceted combinatorial problems can be handled where different subparts are solved efficiently by specialized propagation mechanisms.

* Current address: wh@icparc.ic.ac.uk, IC-Parc, William Penney Laboratory, Imperial College, Exhibition Road, London SW7 2AZ, United Kingdom

An unnerving and not well studied property of propagation based solvers is that the form of a constraint may change the amount of information that propagation discovers. For example consider the constraint $X = Y \wedge X + Y = Z$, and the equivalent constraint $X = Y \wedge 2Y = Z$. Using the second form, the domain of Z (using domain propagation) or at least its upper and lower bounds (using bounds propagation) will be even integers, but this is not necessarily true using the first form. Hence the form of constraints changes the result of the solver. The usual pragmatic choice for representing constraints is to represent them in the form they occur in the program that creates them. This can be improved as the above example shows, and propagation solvers will typically make use of binding constraints $X = Y$ (if they occur before both X and Y occur in another form of constraint) and ground variables to produce more efficient forms of constraints. This still leaves much room for improvement: for example the constraint $X + Y = Z \wedge X = Y$ will not be improved under this policy.

In this paper we investigate the effect of different representations of linear integer constraints in terms of propagation, and show how we can use this information to create efficient bounds propagation solvers. The contributions of the paper are

- A number of theorems which relate the form of a linear constraint to the amount of domain information that can be extracted from the constraint.
- The design and implementation of a bounds propagation solver which makes use of the possibilities of changing the form of linear constraints.
- Empirical comparisons of the solver with `clp(FD)` [3, 5], an efficient indexical solver.

The rest of the paper is organized as follows: in Section 2 we formally define domain and bounds propagation solvers and show a number of different results regarding the amount of domain information they extract from different forms of constraints. In Section 3 we describe the constraint solver we have built to take advantage of some of these results. Then in Section 4 we give an empirical comparison of the solver, showing the effect of various ways of changing the form of the constraints during execution. Finally in Section 5 we conclude.

2. Propagation Based Solving

In this paper we restrict ourselves to linear integer constraint solving. A *primitive constraint* c is a linear equality ($=$), inequality ($\leq$) or disequation ($\neq$), which may be written $\sum_{i=1}^{n} a_i x_i$ op d where a_i, d are integers and op $\in \{=, \leq, \neq\}$. Two particular forms of primitive constraints we shall be interested in are *two variable equations* (TVPE) of the form $a_1 x_1 + a_2 x_2 = d$ where $a_1 \neq 0, a_2 \neq 0$, and *unit coefficient two variable equations* (UTVPE), where $\{a_1, a_2\} \subseteq \{-1, 1\}$.

A *constraint* is a conjunction of primitive constraints, which we will sometimes treat as a set of primitive constraints. An integer (real) valuation θ is a mapping of variables to integer (resp. real) values, written $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$. We extend the valuation θ to map expressions and constraints involving the variables in the natural way. Let *vars* be the function that returns the set of (free) variables appearing in a constraint or valuation. A valuation θ is an integer (real) solution of constraint C if $\mathcal{Z} \models_\theta C$ (resp. $\mathcal{R} \models_\theta C$).

A *range* of integers $[l..u]$ is the set of integers $\{l, l+1, \ldots, u\}$, or $\emptyset$ if $l > u$. A *domain* D is a complete mapping from a fixed (countable) set of variables $\mathcal{V}$ to finite sets of integers. A *false domain* D is a domain where there exists x with $D(x) = \emptyset$. The *intersection* of two domains D_1 and D_2, denoted $D_1 \sqcap D_2$, is defined by the domain $D_3(x) = D_1(x) \cap D_2(x)$ for all x. A domain D_1 is *stronger* than a domain D_2, written $D_1 \sqsubseteq D_2$, if $D_1(x) \subseteq D_2(x)$ for all variables x. A domain D_1 is stronger than (equal to) a domain D_2 w.r.t. variables V, denoted $D_1 \sqsubseteq_V D_2$ (resp. $D_1 =_V D_2$), if for all $x \in V$ $D_1(x) \subseteq D_2(x)$ (resp. $D_1(x) = D_2(x)$). We use the notation $-\{x\}$ to denote the variable set $\mathcal{V} - \{x\}$.

In an abuse of notation, we define a valuation θ to be an element of a domain D, written $\theta \in D$, if $\theta(x_i) \in D(x_i)$ for all $x_i \in vars(\theta)$. We will be interested in determining the infimums and supremums of expressions with respect to some domain D. Define the infimum and supremum of an expression e with respect to a domain D as $\inf_D e = \inf \ \{\theta(e) | \theta \in D\}$ and $\sup_D e = \sup \ \{\theta(e) | \theta \in D\}$.

A propagation solver can be considered as a function $solv(C, D) = D'$ which maps a constraint C and an initial domain D to a new domain $D' \sqsubseteq D$ such that

$$\mathcal{Z} \quad \models \quad \left(C \wedge \bigwedge_{x \in vars(C)} x \in D(x) \right) \leftrightarrow \left(C \wedge \bigwedge_{x \in vars(C)} x \in D'(x) \right)$$

The solver determines that the constraint C and domain D are unsatisfiable when it returns a false domain D'.

A domain D is *domain consistent* with constraint C iff for each primitive constraint $c \in C$ where $vars(c) = \{x_1, \ldots, x_n\}$ then for each $1 \leq i \leq n$ and each value $d_i \in D(x_i)$ there exist $d_j \in D(x_j), 1 \leq j \leq n, j \neq i$ such that $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$ is an integer solution of c. A *domain propagation solver* is such that $solv(C, D)$ is always domain consistent with C.

Define the domain consistency removal function $dc(c, D)$ as returning a domain D' such that when $x_j \notin vars(c)$, $D'(x_j) = D(x_j)$, and when $x_j \in vars(c)$

$$D'(x_j) = \{\theta(x_j) \mid \theta \in D \text{ and } \theta \text{ is a solution of } c\}.$$

A domain propagation solver *dsolv* can be defined as

$$\begin{aligned} diter(C, D) &= \bigsqcap_{c \in C} dc(c, D) \\ dsolv(C, D) &= \mathrm{gfp}(\lambda d.diter(C, d))(D). \end{aligned}$$

EXAMPLE 1 For example consider the constraint $c \equiv x_1 - 3x_2 - 5x_3 = 0$ and the domain $D(x_1) = \{2, 3, 4, 5, 6, 7\}$, $D(x_2) = \{0, 1, 2\}$, $D(x_3) = \{-1, 0, 1, 2\}$. The set of solutions of c in D are $\theta_1 = \{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 0\}$, $\theta_2 = \{x_1 \mapsto 5, x_2 \mapsto 0, x_3 \mapsto 1\}$ and $\theta_3 = \{x_1 \mapsto 6, x_2 \mapsto 2, x_3 \mapsto 0\}$. Hence $dc(c, D)$ gives D' such that $D'(x_1) = \{3, 5, 6\}$, $D'(x_2) = \{0, 1, 2\}$, $D'(x_3) = \{0, 1\}$. D' is domain consistent with respect to c, hence also $dsolv(c, D) = D'$. □

A domain D is *bounds consistent* with constraint C iff for each primitive constraint $c \in C$ where $vars(c) = \{x_1, \ldots, x_n\}$ then for each $1 \leq i \leq n$ and $d_i \in \{\inf_D x_i, \sup_D x_i\}$ there exist *real numbers* $d_j, 1 \leq j \leq n, j \neq i$ where $\inf_D x_j \leq d_j \leq \sup_D x_j$ such that $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$ is a *real* solution of c. A *bounds propagation solver* is such that $solv(C, D)$ is always bounds consistent with C.

Define the bounds consistency removal function $bc(c, D)$ as returning a domain D' such that when $x_j \notin vars(c)$, $D'(x_j) = D(x_j)$, and when $x_j \in vars(c)$ as follows:

- if $c \equiv \sum_{i=1}^n a_i x_i = d$

$$\begin{aligned} D(x_j) \;=\; & \left[\sup\left\{\left\lceil \inf_D \left(\frac{d - \sum_{i=1, i\neq j}^n a_i x_i}{a_j}\right)\right\rceil, \inf_D x_j\right\} \;..\right. \\ & \left. \inf\left\{\left\lfloor \sup_D \left(\frac{d - \sum_{i=1, i\neq j}^n a_i x_i}{a_j}\right)\right\rfloor, \sup_D x_j\right\}\right] \end{aligned}$$

- if $c \equiv \sum_{i=1}^n a_i x_i \leq d$, $a_j > 0$

$$D(x_j) = \left[\inf_D x_j \;..\; \inf\left\{\left\lfloor \frac{d - \sum_{i=1, i\neq j}^n \inf_D(a_i x_i)}{a_j}\right\rfloor, \sup_D x_j\right\}\right]$$

- if $c \equiv \sum_{i=1}^n a_i x_i \leq d$, $a_j < 0$

$$D(x_j) = \left[\sup\left\{\left\lceil \frac{d - \sum_{i=1, i\neq j}^n \inf_D(a_i x_i)}{a_j}\right\rceil, \inf_D x_j\right\} \;..\; \sup_D x_j\right]$$

- if $c \equiv \sum_{i=1}^n a_i x_i \neq d$,

$$D(x_j) = \begin{cases} [\inf_D x_j \;..\; \sup_D x_j] & \exists i \; 1 \leq i \leq n, i \neq j \text{ where } |D(x_i)| \geq 2 \\ [\inf_D x_j + 1 \;..\; \sup_D x_j] & \inf_D x_j = \frac{d - \sum_{i=1, i\neq j}^n a_i d_i}{a_j} \text{ where } D(x_i) = \{d_i\} \\ [\inf_D x_j \;..\; \sup_D x_j - 1] & \sup_D x_j = \frac{d - \sum_{i=1, i\neq j}^n a_i d_i}{a_j} \text{ where } D(x_i) = \{d_i\}. \\ [\inf_D x_j \;..\; \sup_D x_j] & \text{otherwise} \end{cases}$$

The notations $\lfloor e \rfloor$ and $\lceil e \rceil$ respectively denote the greatest integer not greater than e (e rounded down), and the least integer not less than e (e rounded up). A bounds propagation solver *bsolv* can be defined as

$$\begin{aligned} biter(C, D) &= \bigsqcap_{c \in C} bc(c, D) \\ bsolv(C, D) &= \text{gfp}(\lambda d. biter(C, d))(D). \end{aligned}$$

EXAMPLE 2 Consider the same constraint c and domain D as in Example 1. The calculation of $bc(c, D)$ determines that

$$D'(x_1) = \left[\sup\left\{\left\lceil \frac{0 + 3 \times 0 + 5 \times -1}{1}\right\rceil, 2\right\} .. \inf\left\{\left\lfloor \frac{0 + 3 \times 2 + 5 \times 2}{1}\right\rfloor, 7\right\}\right] = [2..7]$$

and

$$D'(x_3) = \left[\sup\left\{\left\lceil\frac{0-3\times 2+1\times 2}{5}\right\rceil, -1\right\} .. \inf\left\{\left\lfloor\frac{0-3\times 0+1\times 7}{5}\right\rfloor, 2\right\}\right] = [0..1]$$

thus while the domain of x_3 is modified, the domain of variable x_1 is unchanged (as is that of x_2). The resulting domain D' is bounds consistent with the constraint c. Notice that bounds propagation has determined less information than domain propagation. □

With a formal definition of a domain propagation solver and a bounds propagation solver we are now in a position to establish results about the behaviour of the solvers on different forms of the same constraint. The following proposition is widely known; effectively it states that the order of applying propagation rules makes no difference to the result. The result is a straightforward application of Cousot's chaotic iteration theorem [4]. We make use of the freedom of computation of $solv(C, D)$ implied by this result in many proofs.

PROPOSITION 1 *Let $C = c_1 \wedge \cdots \wedge c_n$ and let $J_k, k \geq 1$ be an infinite set of subsets of $\{1, \ldots, n\}$ such that for each k and $1 \leq i \leq n$ there exists $l > 0$ such that $i \in J_{k+l}$. When $solv \in \{dsolv, bsolv\}$ and iter is the corresponding iteration function, if $D_0 = D$ and $D_{k+1} = iter(\wedge_{i \in J_k} c_i, D_k)$, then*

$$solv(C, D) = \bigsqcap_{k \geq 1} D_k$$

Next we give two simple lemmas for use in various proofs later on. The first establishes that scaling a linear constraint does not change its propagation behaviour.

LEMMA 1 *Consider the primitive constraints $c_1 \equiv \sum_{i=1}^n a_i x_i$ op d and $c_2 \equiv b \sum_{i=1}^n a_i x_i$ op bd, where op $\in \{=, \leq, \neq\}$ and $b > 0$. Then for any domain D and $solv \in \{dsolv, bsolv\}$ we have*

$$solv(c_1, D) = solv(c_2, D)$$

Proof: For domain propagation, clearly the solutions of c_1 and c_2 are identical, hence $dc(c_1, D) = dc(c_2, D)$. For bounds propagation, the numerator and denominator of all terms are both multiplied by b hence $bc(c_1, D) = bc(c_2, D)$. ■

The second lemma establishes that bounds propagation using an equation or its two corresponding inequalities is equivalent.

LEMMA 2 *Consider the primitive constraint $c_1 \equiv \sum_{i=1}^n a_i x_i = d$ and the two constraints $c_2 \equiv \sum_{i=1}^n a_i x_i \leq d$ and $c_3 \equiv \sum_{i=1}^n -a_i x_i \leq -d$. Then for any domain D we have*

$$bsolv(c_1, D) = bsolv(c_2 \wedge c_3, D)$$

Proof: From the definition of bc we can see that $bc(c_2, D) \sqcap bc(c_3, D) = bc(c_1, D)$. ■

EXAMPLE 3 Note that the same result does not hold for domain propagation. For example the constraint $x_1 - 2x_2 \leq 0 \wedge -x_1 + 2x_2 \leq 0$ with domain $D(x_1) = D(x_2) = \{1, 2, 3, 4\}$ gives domain $D'(x_1) = \{2, 3, 4\}, D'(x_2) = \{1, 2\}$ using either domain or bounds propagation. But the equivalent equation $x_1 - 2x_2 = 0$ leads to the domain $D''(x_1) = \{2, 4\}, D''(x_2) = \{1, 2\}$ using domain propagation. □

The next lemma relates the domains of variables occurring in a two variable equation (TVPE). For these constraints we obtain a one to one correspondence between domain elements or bounds. We shall make use of this correspondence in later results.

LEMMA 3 *Consider the primitive constraint* $c \equiv b_j x_j + b_k x_k = e$, $b_j \neq 0, b_k \neq 0$. *Then for any domain D we have (a) there is a one-to-one correspondence between elements of the domains of x_j and x_k in $dsolv(c, D)$; that is $d_j \in dsolv(c, D)(x_j)$ iff $\frac{e - b_j d_j}{b_k} \in dsolv(c, D)(x_k)$ and (b) there is a one to one correspondence between bounds for x_j and x_k in $D' = bsolv(c, D)$; that is* $\inf_{D'} b_j x_j + \sup_{D'} b_k x_k = e$ *and* $\sup_{D'} b_j x_j + \inf_{D'} b_k x_k = e$.

Proof: Straightforward ■

The following lemma establishes that a ground variable can be eliminated from a constraint without changing propagation behaviour. This justifies the first kind of substitution we use later: replacing ground variables by their values.

LEMMA 4 *Consider the primitive constraint* $c_1 \equiv \sum_{i=1}^{n} a_i x_i$ op d *where* op $\in \{=, \leq, \neq\}$. *Let* $c_2 \equiv \sum_{i=1, i \neq j}^{n} a_i x_i$ op $d - a_j d_j$. *Then for $solv \in \{dsolv, bsolv\}$ and any domain D where $D(x_j) = \{d_j\}$ we have that*

$$solv(c_1, D) = solv(c_2, D)$$

Proof: For domain propagation, clearly any solution $\theta \in D$ for c_1 is also a solution of c_2 and vice versa since $x_j = d_j$ in all such solutions. Hence $dc(c_1, D) = dc(c_2, D)$. For bounds propagation $\inf a_j x_j = \sup a_j x_j = a_j d_j$ and hence $bc(c_1, D) = bc(c_2, D)$. The condition $D(x_j) = \{d_j\}$ either continues to hold or both propagations give false domains. ■

Often large linear constraints are broken up into smaller parts, in order to reduce the kind of constraints to a small number of forms by introducing new variables. A common restriction is to reduce all primitive constraints to involve at most 3 variables. For example the constraint

$$x_1 + x_2 - 2x_3 - 2x_4 = 5$$

can be decomposed into

$$v + x_2 - 2x_4 = 5 \wedge v = x_1 - 2x_3.$$

The next two theorems show that this splitting process, introducing a new variable to represent a linear subterm of the constraint, does not change the domain information determined by domain propagation or bounds propagation.

THEOREM 1 *Consider the primitive constraint* $c_1 \equiv \sum_{i=1}^n a_i x_i \text{ op } d$, *where* op $\in \{=, \leq, \neq\}$. *Let* v *be some variable* $v \notin \{x_1, \ldots, x_n\}$. *Then for any subset* $V \subseteq \{1, \ldots, n\}$ *and domain* D *where* $D(v) \supseteq \{\sum_{i \in V} a_i \theta(x_i) \mid \theta \in D\}$ *we have*

$$dsolv(c_1, D) =_{vars(c_1)} dsolv(c_2 \wedge c_3, D)$$

where $c_2 \equiv v + \sum_{i=1, i \notin V}^n a_i x_i \text{ op } d$ *and* $c_3 \equiv v = \sum_{i \in V} a_i x_i$.

Proof: We show that any value $d_j \in dsolv(c_1, D)(x_j)$ must appear in $dsolv(c_2 \wedge c_3, D)(x_j)$ and vice versa. Let $D' = dsolv(c_1, D)$. Then by the definition of dc there exists $\theta \in D'$ where $\theta(x_j) = d_j$ and θ is a solution of c_1. Extend θ to θ' where $\theta'(x_i) = \theta(x_i), 1 \leq i \leq n$ and $\theta'(v) = \sum_{i \in V} a_i \theta(x_i)$. Clearly θ' is a solution of $c_2 \wedge c_3$, and because $\theta \in D$ and $D(v)$ is large enough we have that $\theta' \in D$.

Suppose to the contrary that $\theta' \notin dsolv(c_2 \wedge c_3, D)$. Then there must exist a first step in the fixpoint calculation where the domain D'' is such that $\theta' \in D''$ but $\theta' \notin diter(c_2 \wedge c_3, D'')$. Choose some k such that $d_k \in D''(x_k) - diter(c_2 \wedge c_3, D'')(x_k)$. Then there cannot exist a solution $\phi \in D''$ of c_2 or c_3 that has $\phi(x_k) = d_k$. But θ' is such a solution. Contradiction. Thus $\theta' \in dsolv(c_2 \wedge c_3, D)$.

The reverse direction follows similarly. ■

THEOREM 2 *Consider the primitive constraint* $c_1 \equiv \sum_{i=1}^n a_i x_i \text{ op } d$, *where* op $\in \{=, \leq, \neq\}$. *Let* v *be some variable* $v \notin \{x_1, \ldots, x_n\}$. *Then for any subset* $V \subseteq \{1, \ldots, n\}$ *and domain* D *where* $D(v) \supseteq [\sum_{i \in V} \inf_D a_i x_i .. \sum_{i \in V} \sup_D a_i x_i]$, *we have*

$$bsolv(c_1, D) =_{vars(c_1)} bsolv(c_2 \wedge c_3, D)$$

where $c_2 \equiv v + \sum_{i=1, i \notin V}^n a_i x_i \text{ op } d$ *and* $c_3 \equiv v = \sum_{i \in V} a_i x_i$.

Proof: See the appendix. ■

Since breaking up a primitive constraint into two parts by introducing a new variable does not affect the result of propagation, it is interesting to consider when we can perform the reverse operation, substitution, and improve the strength of propagation. We shall concentrate on substitution using two variable equations (TVPEs).

Definition 1. Let $c_1 \equiv \sum_{i=1}^n a_i x_i \text{ op } d$, where op $\in \{=, \leq, \neq\}$, and let $c_2 \equiv b_j x_j + b_k x_k = e, j \neq k, b_j > 0, b_k \neq 0$. Then the *substitution* of x_j using c_2 applied to c_1, denoted $subs(c_1, x_j, c_2)$ is defined as

$$subs(c_1, x_j, c_2) \equiv (b_j a_k - a_j b_k) x_k + b_j \sum_{i=1, i \notin \{j,k\}}^n a_i x_i \text{ op } b_j d - a_j e.$$

When $b_j < 0$ we define $subs(c_1, x_j, c_2) = subs(c_1, x_j, -b_j x_j - b_k x_k = -e)$.

The following theorems show the effects of substitution using a two variable equation. The first theorem shows that substitution by a two variable equation can never weaken the domain propagation that occurs.

THEOREM 3 *Consider the primitive constraints* $c_1 \equiv \sum_{i=1}^{n} a_i x_i$ op d, *where* op $\in \{=, \leq, \neq\}$, *and* $c_2 \equiv b_j x_j + b_k x_k = e, j \neq k, b_j \neq 0, b_k \neq 0$. *Let* $c_3 = subs(c_1, x_j, c_2)$. *Then for all domains* D *we have*

$$dsolv(c_1 \wedge c_2, D) \sqsupseteq dsolv(c_3 \wedge c_2, D)$$

Proof: The case for $a_j = 0$ follows from Lemma 1, so we assume $a_j \neq 0$. Let $D' = dsolv(c_1 \wedge c_2, D)$ and $D'' = dsolv(c_3 \wedge c_2, D)$. Consider an arbitrary solution $\theta \in D''$ of c_3. Define θ' as $\theta'(x) = \theta(x), x \in vars(c_3)$ and $\theta'(x_j) = (e - b_k\theta(x_k))/b_j$. Then by Lemma 3, $\theta' \in D''$ and hence $\theta' \in D$. Now θ' is a solution of c_2 and because $c_1 \wedge c_2$ has the same solutions as $c_3 \wedge c_2$ it is also a solution of c_1.

Suppose that $\theta' \notin D'$. Then there exists an iteration in the calculation of $dsolv(c_1 \wedge c_2, D)$ such that $\theta' \in D_i$ and $\theta' \notin D_{i+1} = diter(c_1 \wedge c_2, D_i)$. But by the definition of dc and the fact that θ' is a solution of c_1 and c_2 this cannot occur. This yields a contradiction, and so $\theta' \in D'$. For every variable x_i, every domain element in $D''(x_i)$ appears in some solution of $c_2 \wedge c_3$, and so we have that $D'' \sqsubseteq D'$.
■

EXAMPLE 4 The result of substituting a two variable equation can be strictly stronger. Consider the constraints

$$\begin{aligned} c_1 &\equiv x_1 + x_2 - 2x_3 - 2x_4 = 5 \\ c_2 &\equiv x_1 - x_2 = 0 \end{aligned}$$

and the domain $D(x_1) = D(x_2) = \{0, 1, 2, 3, 4, 5\}, D(x_3) = D(x_4) = \{0, 1, 2\}$. $dsolv(c_1 \wedge c_2, D) = D$. Substituting for x_1 in c_1 using c_2 we obtain

$$c_3 \equiv 2x_2 - 2x_3 - 2x_4 = 5$$

but $dsolv(c_3 \wedge c_2, D)$ returns a false domain, since there can be no solutions to c_3.
□

We can in fact characterize when we may get benefits to domain propagation by substitution. The next theorem defines (negatively) the cases when substitution can lead to improved domain propagation.

THEOREM 4 *Consider the primitive constraints* $c_1 \equiv \sum_{i=1}^{n} a_i x_i$ op d *where* op $\in \{=, \leq, \neq\}$, *and* $c_2 \equiv b_j x_j + b_k x_k = e, j \neq k, b_j \neq 0, b_k \neq 0$. *Let* $c_3 = subs(c_1, x_j, c_2)$. *Then*

$$dsolv(c_1 \wedge c_2, D) = dsolv(c_3 \wedge c_2, D)$$

for all domains D *iff*

(a) op $\in \{=, \neq, \leq\}$ *and* $a_j = 0$ *or* $a_k = 0$;

(b) $n = 2$, op $\in \{\leq\}$ *and* $a_j b_j a_k b_k > 0$;

(c) $n = 2$, op $\in \{=\}$ *and* $|a_j b_k / b_j a_k| \neq 1$;

(d) $dsolv(c_1 \wedge c_2, D)$ *gives a false domain for all domains* D*; or*

(e) $c_3 \leftrightarrow$ *true.*

Proof: See the appendix. ■

As a consequence of the above theorem we can see there may be disadvantages with splitting constraints into parts: it may remove the advantages of substitution.

EXAMPLE 5 Considering the previous example, the same propagation behaviour results from replacing c_1 by

$$v + x_2 - 2x_4 = 5 \wedge v = x_1 - 2x_3,$$

but then if we use c_2 to substitute for x_1 we obtain

$$v + x_2 - 2x_4 = 5 \wedge v = x_2 - 2x_3 \wedge x_1 - x_2 = 0.$$

We are guaranteed by Theorem 4 that propagation is never improved by this substitution. Hence we obtain weaker propagation behaviour than if we had not split c_1 before substitution $(2x_2 - 2x_3 - 2x_4 = 5 \wedge x_1 - x_2 = 0)$. □

For bounds propagation we have a similar result to domain propagation: substitution by a two variable equation can only improve bounds found in the resulting propagation.

THEOREM 5 *Consider the primitive constraints* $c_1 \equiv \sum_{i=1}^{n} a_i x_i$ op d *where* op $\in \{=, \leq, \neq\}$, *and* $c_2 \equiv b_j x_j + b_k x_k = e, j \neq k$, $b_j \neq 0$, $b_k \neq 0$. *Let* $c_3 = subs(c_1, x_j, c_2)$. *Then for all domains* D *we have*

$$bsolv(c_1 \wedge c_2, D) \sqsupseteq bsolv(c_3 \wedge c_2, D)$$

Proof: See the appendix. ■

Given we can get stronger domain information by substituting using a two variable equation, it is natural to ask whether we can use a two variable equation to totally eliminate a variable from consideration. The next theorems show when this elimination is guaranteed not to lose information.

THEOREM 6 *Given a constraint* C *and two variable equation* $c_2 \equiv b_j x_j + b_k x_k = e, j \neq k$, $b_j \neq 0$, $b_k \neq 0$. *Let* $C' = \wedge_{c \in C} subs(c, x_j, c_2)$. *Then for all domains* D *we have*

$$dsolv(C \wedge c_2, D) \sqsupseteq_{-\{x_j\}} dsolv(C', dsolv(c_2, D)).$$

Proof: By Lemma 3 in $dsolv(c_2, D)$ there is a one to one correspondence between values for x_j and x_k and hence for $D' = dsolv(C', dsolv(c_2, D))$ we have $dsolv(c_2, D') =_{-\{x_j\}} D'$ since no $d_k \in D'(x_k)$ will be removed (x_j's corresponding value will be in $D'(x_j)$). The result then follows from Proposition 1 and Theorem 3. ■

THEOREM 7 *Given a constraint C and two variable equation $c_2 \equiv b_j x_j + b_k x_k = e, j \neq k$, $b_j = \pm 1$, $b_k \neq 0$. Let $C' = \wedge_{c \in C} subs(c, x_j, c_2)$. Then for all domains D we have*

$$bsolv(C \wedge c_2, D) \sqsupseteq_{-\{x_j\}} bsolv(C', bsolv(c_2, D)).$$

Proof: By Lemma 3 in $bsolv(c_2, D)$ there is a correspondence between bounds for x_j and x_k and again for $D' = bsolv(C', bsolv(c_2, D))$ we have $bsolv(c_2, D') =_{-\{x_j\}} D'$ because since $b_j = \pm 1$ there is no rounding possible for x_j and hence no $d_k \in D'(x_k)$ will be removed. The result then follows from Proposition 1 and Theorem 5. ■

EXAMPLE 6 Note that if $b_j \neq \pm 1$, then performing a two variable equation elimination may result in weaker bounds propagation. Consider the following example. Let

$$\begin{aligned} c_1 &\equiv 2x_1 + x_2 = 3 \\ c_2 &\equiv 2x_1 + x_3 = 4 \\ c_3 = subs(c_1, x_1, c_2) &\equiv x_2 - x_3 = -1. \end{aligned}$$

Let D be the domain

$$D(x_1) = [0..2], D(x_2) = [0..4], D(x_3) = [0..4].$$

Now if $D' = bsolv(c_1 \wedge c_2, D)$ and $D'' = bsolv(c_3, bsolv(c_2, D))$ (note that $bsolv(c_2, D) = D$), then

$$D'(x_1) = [0..1], D'(x_2) = [1..3], D'(x_3) = [2..4]$$

while

$$D''(x_1) = [0..2], D''(x_2) = [0..3], D''(x_3) = [1..4].$$

Thus the original version is stronger than the transformed: the information that x_3 must be even and x_2 must be odd has been lost. Note that domain propagation eliminates all the odd elements of x_3 when propagating through c_2, and thus does not suffer from this problem. □

We conclude this section by remarking that almost no substitution using a three (or more) variable equation will result in propagation which can be guaranteed to be no weaker than the original. This is due to there being no analogue of Lemma 3 for equations with more than two variables. As a result, the domain or bounds corresponding to the expression replacing the substituted variable will often be weaker than those of the variable itself, resulting in weaker propagation on the other variables in the constraint.

EXAMPLE 7 Consider the following example

$$\begin{aligned} c_1 &\equiv x_1 + x_2 + x_3 - x_4 = 5 \\ c_2 &\equiv x_3 + x_4 + x_5 = 3 \\ c_3 &\equiv x_1 + x_2 - 2x_4 - x_5 = 2 \end{aligned}$$

Here, x_3 in c_1 has been replaced using c_2 by $3 - x_4 - x_5$ in c_3. Suppose we are working with the domain D such that

$$D(x_1) = D(x_2) = [0..15], D(x_3) = D(x_4) = D(x_5) = [0..3].$$

Note that $bsolv(c_2, D) = D$. Let $D' = solv(c_1 \wedge c_2, D)$ and $D'' = solv(c_3 \wedge c_2, D)$. Now, x_3 has domain $[0..3]$, but the expression $3 - x_4 - x_5$ has domain $[-3..3]$. As a result, while $D'(x_1) = D'(x_2) = [0..8]$, we have $D''(x_1) = D''(x_2) = [0..11]$, which is weaker (the domains of x_3, x_4 and x_5 are unchanged in both cases). Note that for the same constraints but a different domain the reverse relationship can hold. Consider D defined as

$$D(x_1) = D(x_2) = [0..10], D(x_4) = [-9..9], D(x_3) = D(x_5) = [-3..6]$$

(again, $bsolv(c_2, D) = D$). Then we have $D'(x_4) = [-8..9]$, while $D''(x_4) = [-4..9]$. □

3. A Tableau Based Bounds Propagation Solver

Indexicals were introduced in [12] (published later as [13]) as a mechanism for specifying propagation behaviour. Indexicals have been widely used for implementing finite domain solvers, including that of the `clp(FD)` compiler [3, 5]. The idea behind indexicals is that rather than implementing support for a large number of primitive constraints directly, and optimising each separately and in combination, the compiler should only implement a few low-level primitives (indexicals) and implement them well.

The indexical approach builds n indexicals for a n-variable constraint. When one of the variables in an n-variable constraint is modified, $n-1$ indexicals are woken for propagation. The indexicals woken will often share some redundant computation, which is repeated for each woken indexical. This problem has also been noted in [3], where the solution presented is to break up constraints with many terms into several smaller ones, by introducing new intermediate variables. We have seen in Theorem 1 and Theorem 2 that this breakup of constraints does not reduce the amount of domain information discovered. The breakup reduces the amount of redundant computation, and can in some cases reduce the number of iterations of propagation required.

As a result of Lemma 4, Theorem 5 and Theorem 7 we are interested in being able to manipulate the form of constraints in a bounds propagation solver. In particular we wish to substitute ground variables out of constraints, and substitute variables appearing in unit two variable equations. This is problematic using an indexical representation since each primitive constraint is compiled into multiple indexicals. As a result, we propose an alternative representation of the constraints, based on a constraint tableau.

Given Theorem 4 and the subsequent example, we can see there are disadvantages to splitting primitive constraints when substitutions may be performed. Splitting a primitive constraint may prevent any gain in propagation behaviour. Hence for our alternate approach we will avoid the splitting of primitive constraints, in order to make substitution more effective. But since we deal with a primitive constraint as a whole the redundant computation created by indexicals will not occur in our approach.

3.1. Constraint Representation

The representation we use for our new implementation of a propagation solver is similar to that used in CLP($\mathcal{R}$) [6] for representing real constraints. Essentially, each primitive constraint is represented as a list of terms appearing in the constraint, plus a constant. Each primitive constraint is tagged with its type (equation, inequality, disequation), and a count of the number of terms appearing in it. Each node in a term list stores the value of the coefficient, the ID of the variable, a pointer to the next term, and a pointer to the parent constraint node.

A variable node stores the lower and upper bounds of the variable as well as a doubly linked list of excluded values (holes in the domain). In addition there are four lists used for propagation, containing the term node occurrences of the variable in: inequalities where it has a positive coefficient, inequalities where it has a negative coefficient, equations, and disequations. Each term node stores a pointer to the next occurrence of the same variable in the same context (for example, in a disequation, or in an inequality with a negative coefficient). The four lists contain all the primitive constraints which contain the variable. They also respectively hold the primitive constraints that need to be propagated whenever the variable: has its lower bound updated, has its upper bound updated, has either its lower or upper bound updated, and becomes ground.

An example of what the data structures for the inequality $3X - 2Y \leq 1$ might look like appears in Figure 1.

The primitive constraint itself is used as a propagation rule for all the variables in the constraint, as opposed to indexicals which update only one variable. This means that new bounds on any number of the variables appearing in the primitive constraint can be propagated to all other variables in only two passes over the primitive constraint. Hence propagation is linear in the number of variables in the constraint, compared to quadratic for the indexical scheme. It is also reasonably straightforward to manipulate the constraints if desired, for example substituting out a variable which has become ground, or performing substitutions of one variable by another.

Let $c \equiv \sum_{i=1}^{n} a_i x_i \leq d$. Calculations performed in evaluating $bc(c, D)$ share a great deal of work. Given $a_j > 0$ we calculate the (possibly new) upper bound for x_j

$$u_j = \left\lfloor \frac{d - \sum_{i=1, i \neq j}^{n} \inf_D a_i x_i}{a_j} \right\rfloor$$

If we let $S = d - \sum_{i=1}^{n} \inf_D a_i x_i$ then we can alternatively calculate this as

$$\left\lfloor \frac{S + \inf_D a_j x_j}{a_j} \right\rfloor = \left\lfloor \frac{S + a_j \inf_D x_j}{a_j} \right\rfloor$$

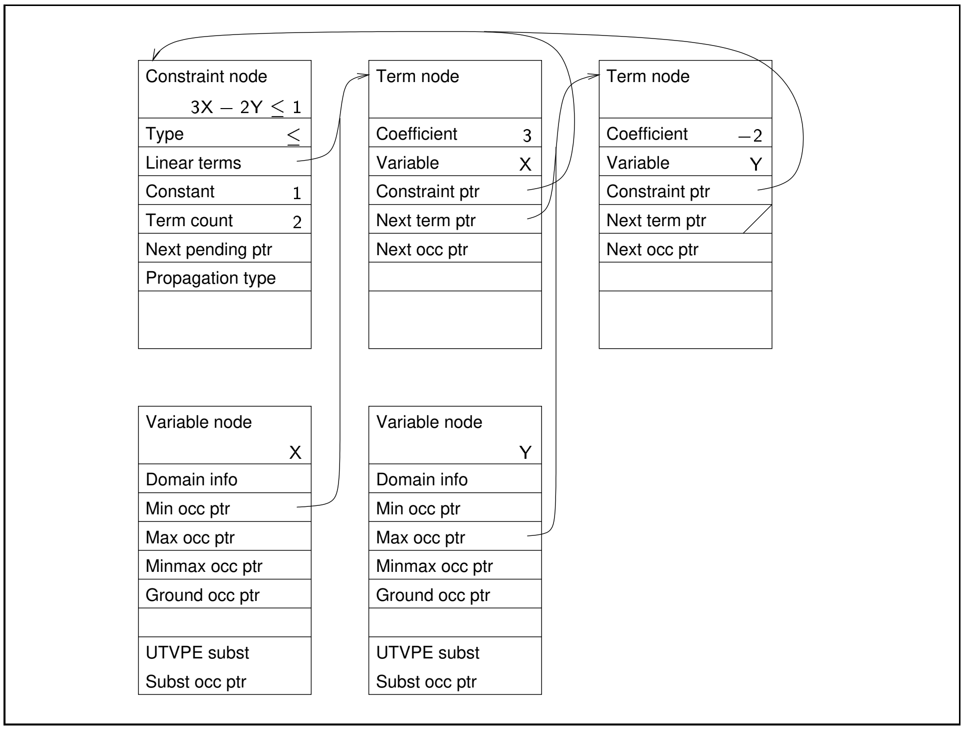

Figure 1. Example constraint representation

Similarly when $a_j < 0$ the (possibly new) lower bound is

$$\left\lceil \frac{S + \inf_D a_j x_j}{a_j} \right\rceil = \left\lceil \frac{S + a_j \sup_D x_j}{a_j} \right\rceil$$

Thus we can determine all the new bounds in two passes over the constraint: one to compute S, the other to compute the bounds.

For the purposes of computing bounds, equations can be considered as a pair of inequalities (see Lemma 2). Propagation using disequations only occurs when all but one variable is ground and then they (possibly) remove an element from the domain of this variable (possibly causing unsatisfiability).

3.2. The Propagation Queue

For indexical solvers the propagation queue contains the indexicals that are waiting to be propagated. In the tableau approach the queue contains primitive constraints waiting to be propagated through. If a primitive constraint is in the propagation queue, it is labelled with the type of the pending propagation and a pointer to the next constraint in the queue. The propagation type is used for testing membership of the propagation queue, as well as for distinguishing which "half" (or both) of an equation needs propagating.

Each time a variable's bound is modified, all the primitive constraints which then need to be propagated are added to the queue. This is achieved by traversing the appropriate occurrence lists and adding the constraints there to the propagation queue, or for an equation possibly modifying which "halves" require propagation. A standard optimisation to avoid redundant work is to not re-add any constraint already in the queue.

Propagating through an inequality cannot result in waking up the same inequality; however, this is not the case for equations. Despite this, in many situations propagating back through the same equality constraint can be guaranteed to give no new information. The following theorem characterises some cases where re-propagation is not necessary. We assume here that the equation is modelled as a pair of inequalities (cf. Lemma 2), and that the solver (as happens in practice) alternates between propagating each inequality.

THEOREM 8 *Let $c_1 \equiv \sum_{i=1}^{n} a_i x_i = d$, with corresponding inequalities $c_2 \equiv \sum_{i=1}^{n} a_i x_i \leq d$ and $c_3 \equiv -\sum_{i=1}^{n} a_i x_i \leq -d$. Consider evaluating $bc(c_2, D)$ (half of the computation for $bc(c_1, D)$). Let α be the number of tighter bounds created which involved integer rounding. Let β be the number of bounds created which are at least as tight as the existing bounds and did not involve integer rounding. Then assuming that $bc(c_3, D) = D$*

- *if $\alpha = 0$ we have $bc(c_2, D) = bc(c_3, bc(c_2, D))$;*
- *if $\beta \geq 2$ we have $bc(c_2, D) = bc(c_3, bc(c_2, D))$; and*
- *if $\beta = 1$ (bounding on variable x_j) then $bc(c_2, D) =_{-\{x_j\}} bc(c_3, bc(c_2, D))$.*

Proof: Let $D_1 = bc(c_2, D)$ and $D_2 = bc(c_3, D_1)$. Without loss of generality, suppose $a_i > 0$ for all i. The rule for updating the upper bound of x_j using c_2 is then

$$\sup_{D_1} x_j = \sup \left\{ \left\lfloor \frac{c - \sum_{i=1, i \neq j}^{n} a_i \inf_D x_i}{a_j} \right\rfloor, \sup_D x_j \right\}$$

Suppose no rounding occurs when computing this bound during a propagation pass over the constraint, and that the new bound is at least as tight as the existing bound. Then we have (after propagation):

$$\sup_{D_1} x_j = \frac{c - \sum_{i=1, i \neq j}^{n} a_i \inf_{D_1} x_i}{a_j} \tag{1}$$

since $\inf_D x_i = \inf_{D_1} x_i$ for all $1 \leq i \leq n$. Consider a different variable, x_k, appearing in the constraint. To update this variable using c_3 based on the new upper bounds deduced during the propagation pass that updated x_j, we have

$$\inf_{D_2} x_k \geq \frac{c - \sum_{i=1, i \neq k}^{n} a_i \sup_{D_1} x_i}{a_k}$$

In particular, making use of (1), we have

$$\begin{aligned}
\inf_{D_2} x_k &\geq \frac{c - \sum_{i=1,i\neq j,i\neq k}^{n} a_i \sup_{D_1} x_i - c + \sum_{i=1,i\neq j}^{n} a_i \inf_{D_1} x_i}{a_k} \\
&= \frac{a_k \inf_{D_1} x_k - \sum_{i=1,i\neq j,i\neq k}^{n} a_i (\sup_{D_1} x_i - \inf_{D_1} x_i)}{a_k} \\
&= \inf_{D_1} x_k - \frac{\sum_{i=1,i\neq j,i\neq k}^{n} a_i (\sup_{D_1} x_i - \inf_{D_1} x_i)}{a_k}
\end{aligned}$$

Since $\sup_{D_1} x_i - \inf_{D_1} x_i \geq 0$ (or we would have already detected unsatisfiability), it is clear that the RHS cannot be larger than the $\inf_{D_1} x_k$, and thus there is no way for such a propagation to yield a new lower bound for x_k.

Two cases follow immediately. If (1) holds for exactly one variable x_j then for all $k \neq j$ we can guarantee no new bound (third case). If (1) holds for more than one value of j, then clearly we can guarantee no new lower bound for all k (second case).

Suppose $\alpha = 0$. Let β' be the number of deduced bounds which did not involve integer rounding and are strictly tighter than the existing bounds (so $\beta' \leq \beta$). We can assume $\beta' < 2$, since otherwise we can just use the previous case. If $\beta' = 0$, then clearly none of the upper bounds were modified, and so propagation of these bounds is useless. If $\beta' = 1$, only one upper bound has been modified, and by the third case, only the corresponding lower bound need be considered for update. However, the rule for updating this lower bound is independent of the (only) modified upper bound, and so no update can occur. Thus no updates can occur for any value of β', and the first case holds. ■

Note that the $bc(c_3, D) = D$ condition is often known to be true without an explicit check. For instance, it is true if we have just propagated through c_3, or if none of the relevant bounds have been modified since the last time such a propagation occurred.

Indexical solvers typically handle the possible repropagation of equations by compiling equations with all unit (± 1) coefficients specially. In this case it is guaranteed that α will always be 0. Hence any indexicals derived from the same equation need not be repropagated. This kind of optimisation is important for the indexical representation, where a primitive constraint generates n rules, $n-1$ of which would otherwise be woken.

We chose a different approach, which is facilitated by the tableau representation. We repeatedly propagate through an equation until $\alpha = 0$ or $\beta \geq 2$, before doing any other propagation. This may avoid propagating useless bounds elsewhere, but may also do unnecessary work in some cases. Empirical comparisons showed a slight benefit in repropagating immediately. Note that we need to calculate α at run-time unlike an indexical solver, since, due to substitutions, coefficients can change during the computation.

3.3. Variable Substitution

We have identified two kinds of substitution which never reduce the amount of bounds information determined by the solver: substitution of ground variables and substitution of a variable with a unit coefficient in a two variable equation.

The first kind of substitution makes the constraints smaller (fewer variables), thus speeding up their propagation. We can choose to substitute ground variables out of new primitive constraints only, or out of all existing primitive constraints as well. The first case has an extremely low overhead and thus least likely to have a negative impact; one simply adjusts the constant term for the primitive constraint being generated rather than adding a new variable term. The second case involves manipulating every existing constraint containing the variable, which obviously has a higher overhead, and thus a greater chance of impacting performance negatively. However, it also occurs much more often (typically all variables become ground by the end of a successful computation) and thus has a larger scope for improving performance. The computational results presented in Section 4 indeed show that performing the substitutions often yields a significant improvement in execution time, and only occasionally results in a slight penalty.

Performing these ground substitutions of a variable is straightforward. One simply traverses all the occurrence lists for a variable, deleting the relevant term node from each primitive constraint, and updating the constants (if the primitive constraint is now completely ground, we may detect failure). If a substitution causes a primitive constraint to become of a special form (single variable or unit coefficient two variable equation (UTVPE)), then it is marked for further processing after the substitution is complete. Any pending propagation then continues as before (the modified primitive constraints will already be in the propagation queue if they need to be there).

Substituting using an equation is more complex. For simplicity, we only substitute using a two variable equation where both coefficients are unit (± 1) (a UTVPE constraint). These are by far the most common kind of equation occurring in finite domain programs (TVPE constraints with non-unit coefficients did not occur in any of the problems studied). If UTVPE substitutions are being performed, some extra information is stored in the variable node. If the variable has been eliminated, the substitution is recorded here. If the variable has been used in eliminating other variables, a list of all the substitutions it occurs in is maintained (so that if this variable is subsequently substituted, those substitutions can be updated).

UTVPE substitution removes a variable from the propagation solver, requiring fewer variables to be propagated on; it also never increases the size of any constraints. It can be applied using only primitive constraints that are UTVPE when they reach the solver (possibly after ground variables have been substituted out), or in addition using constraints that become UTVPE as the computation progresses (through ground or UTVPE substitution). Since we derive maximum benefit from UTVPE substitution if we completely eliminate a variable from the solver, we perform the substitution on all primitive constraints currently in the solver, as well as any new ones added. As a result, the overheads are much higher than for ground substitutions, and more propagation needs to be saved or pruning achieved to yield a benefit.

To perform a UTVPE substitution $X = aY + c$, we traverse all the occurrence lists for X and in each primitive constraint replace it by Y, updating the variable coefficient and constant term as necessary. If Y already appears in the constraint, then the two terms need to be summed (and both deleted if they sum to zero). Note that if the constraint is an inequality and Y already appears, care needs to be taken that the summed node ends up in the correct occurrence list (min or max). Note also that when adding X's occurrence lists to Y's, the min and max lists need to be exchanged if a is negative. The final thing that needs to be done for a UTVPE substitution is to propagate X's domain information to Y.

Backtracking may cause substitutions to be undone. This is fairly straightforward; basically it amounts to a reverse substitution. This is made easy by not deleting the occurrence lists of substituted variables, which then form the basis for the reverse substitution.

4. Computational Results

The tableau based solver is embedded in the CLP($\mathcal{R}$) interpreter, replacing the usual solver. We compare a number of different variants of the new solver against each other, and against `clp(FD)` [5]. The variants of the new propagation solver are:

`base` Basic version: ground variables are substituted when a new constraint is added to the solver, but no other substitution is performed.

`eqn` Like `base`, but equation optimisation is also performed ($\alpha = 0$ case only).

`ground` Like `eqn`, but variables which become ground are substituted out of all constraints in the store.

`utvpe` Like `ground`, but input UTVPE constraints are used to substitute variables in all constraints in the store as well.

`edetect` Like `utvpe`, but constraints which become UTVPE due to substitutions are also used for substitution.

`clp-fd` The `clp(FD)` compiler, version 2.21. It is based on the `wamcc` Prolog compiler, and thus compiles user code to C before compiling the C code to create an executable.

The benchmarks were written in such a way as to minimise any algorithmic differences between the benchmark code for each system (in order to try to compare just the solvers). In particular, this means that heuristics such as first-fail have not been used. This is because different solvers are likely to select different variables to label first, resulting in the systems traversing different trees and solving different sets of constraints, degrading the usefulness of the comparison.

The benchmarks were run on a 167MHz Sun UltraSPARC Enterprise 1. The numbers presented for each benchmark are the minimum times (in seconds) recorded for 20 separate runs.

The first test problems used are scheduling problems (sched-*), with precedence constraints and resource contention. For each problem we computed worst case bounds a priori (the sum of all task durations) and used these as bounds for all variables. sched-bridge-good uses data from the bridge construction example in [11], and searches for a (modestly) good solution. The remainder are job shop scheduling benchmarks, searching for a first solution. These were taken from J.E. Beasley's OR-Library, from the file `jobshop1.txt`. Since the comparative characteristics were quite similar across all the job shop scheduling benchmarks we tried, we have just selected a few typical ones to present here. The sched-orb* instances are attributed to [2], and the sched-la* to [7].

The second problem set is Fibonacci programs and goals. fib-std-* is the standard (naïve) formulation, with the two recursive calls last. fib-utvpe-* is a slightly reordered formulation such that all the equations are UTVPE (after initial ground substitution) at the time they are passed to the solver. Both were run in the forward (in, out) and backward (out, in) directions, for two different Fibonacci numbers (14/610 and 18/4181).

temp-* are some temporal reasoning problems, taken from [9], using the "shared-track" knowledge base (temp-st-*; the times shown are for 10 repeats). hamil-* is a program for calculating Hamiltonian paths described in [8]. The benchmark was run on different data sets, ranging in size from 10 nodes to 50 nodes, searching for a first solution. Entries in the table of the form "?" indicate that the benchmark took in excess of 8 hours to run.

queens-* is the well-known "N Queens" benchmark. We evaluate two versions: the *forward checking* (queens-fc-*) version, and the more usual *generalised forward checking* (queens-gfc-*) version (see [11] for details), searching for all solutions using board sizes of 8, 10 and 12.

The next set are small combinatorial problems. magic is a (3x3) magic squares problem (all solutions); zebra is the well known "Who owns the zebra?" problem; crypt* are crypt-arithmetic problems from [11] (crypta is send-more-money), using an explicit carry. (The times shown for these benchmarks are for 100 repeats.)

bramsey is a boolean formulation of the problem of finding a 3-colouring of a complete graph with N vertices such that no monochrome triangles exist (note that 16 is the largest N for which a solution exists). perm is the permutation generation example from [11], for which we generated a number of random problems (using the same technique as in [11]). Each eq-* problem is a set of N linear equations on seven variables with domains [0..10], searching for all solutions (though there is only one); the times shown are for 10 repeats.

The results (in terms of CPU time) appear in Table 1. Table 2 shows the number of failures[1] recorded during the execution of each problem for each variant of our solver, where "=" indicates the same as `base`. (Note that, as expected, the number of fails for `eqn` and `ground` are the same as for `base`. This is because these optimi-

Table 1. Execution times

	base	eqn	ground	utvpe	edetect	clp_fd
sched-bridge-good	1.26	1.27	1.33	1.25	1.23	1.02
sched-orb03	0.23	0.24	0.24	0.23	0.23	0.18
sched-orb06	0.24	0.24	0.25	0.23	0.23	0.19
sched-la18	0.22	0.24	0.25	0.24	0.21	0.19
sched-la21	0.74	0.74	0.77	0.73	0.71	0.60
fib-utvpe-forw-14	0.02	0.01	0.02	0.01	0.01	0.02
fib-std-forw-14	0.04	0.04	0.02	0.02	0.02	0.03
fib-utvpe-back-14	0.07	0.07	0.05	0.03	0.02	0.04
fib-std-back-14	0.10	0.10	0.07	0.08	0.07	0.07
fib-utvpe-forw-18	0.19	0.16	0.15	0.07	0.07	0.12
fib-std-forw-18	0.32	0.28	0.24	0.24	0.21	0.26
fib-utvpe-back-18	0.59	0.52	0.48	0.26	0.24	0.31
fib-std-back-18	0.86	0.76	0.66	0.69	0.59	0.61
temp-st-sfenabled	0.10	0.10	0.09	0.10	0.09	0.03
temp-st-sfexits	0.50	0.51	0.49	0.50	0.49	0.11
temp-st-path	0.85	0.87	0.87	0.84	0.84	0.18
hamil-10	0.00	0.00	0.00	0.00	0.00	0.00
hamil-20	15.66	13.23	12.63	0.00	0.00	8.34
hamil-30	?	?	?	0.18	0.19	?
hamil-40	?	?	?	0.98	1.00	?
hamil-50	?	?	?	533.21	544.17	?
queens-fc-8	0.09	0.09	0.09	0.09	0.08	0.24
queens-fc-10	1.64	1.67	1.69	1.69	1.71	5.08
queens-fc-12	40.88	41.24	41.06	41.71	42.22	142.63
queens-gfc-8	0.05	0.05	0.05	0.05	0.04	0.13
queens-gfc-10	0.99	0.97	0.99	1.00	0.99	2.86
queens-gfc-12	25.17	25.17	25.15	25.30	25.15	81.43
magic	0.97	0.88	0.77	0.75	0.71	0.59
zebra	0.11	0.11	0.10	0.08	0.08	0.06
crypta	0.09	0.09	0.07	0.07	0.06	0.09
cryptb	0.37	0.34	0.27	0.26	0.25	0.33
bramsey-12	0.05	0.05	0.05	0.05	0.05	0.04
bramsey-14	1.14	1.08	0.96	0.98	1.32	1.43
bramsey-16	872.84	835.19	762.70	764.83	1053.36	1368.52
perm-11a	0.29	0.26	0.27	0.17	0.19	0.18
perm-11b	0.69	0.66	0.63	0.51	0.50	0.33
perm-11c	1.10	1.04	1.02	0.82	0.82	0.50
perm-12a	4.43	4.15	3.97	3.04	3.07	2.11
perm-12b	4.11	3.90	3.84	3.02	3.03	1.98
perm-12c	7.06	6.61	6.92	5.50	5.46	3.27
eq-10	0.45	0.46	0.34	0.34	0.35	0.44
eq-20	0.50	0.50	0.41	0.42	0.41	0.50

sations are aimed only at improving the efficiency of propagation: they never infer any extra domain information and hence cannot prune the search space.)

Looking at the tables, the optimization of equation handling (`eqn` vs. `base`) seems worthwhile, with modest improvements to most of the benchmarks containing equations, and never causing any significant slow down.

Table 2. Number of failures

	base	eqn	ground	utvpe	edetect
sched-bridge-good	121519	=	=	=	=
sched-orb03	25025	=	=	=	=
sched-orb06	25025	=	=	=	=
sched-la18	25025	=	=	=	=
sched-la21	74600	=	=	=	=
fib-utvpe-forw-14	0	=	=	=	=
fib-std-forw-14	0	=	=	=	=
fib-utvpe-back-14	986	=	=	=	=
fib-std-back-14	986	=	=	=	=
fib-utvpe-forw-18	0	=	=	=	=
fib-std-forw-18	0	=	=	=	=
fib-utvpe-back-18	6764	=	=	=	=
fib-std-back-18	6764	=	=	=	=
temp-st-sfenabled	1286	=	=	=	=
temp-st-sfexits	19800	=	=	=	=
temp-st-path	22211	=	=	=	=
hamil-10	42	=	=	=	=
hamil-20	62219	=	=	53	53
hamil-30	?	?	?	5940	5940
hamil-40	?	?	?	19734	19734
hamil-50	?	?	?	8361422	8361422
queens-fc-8	7278	=	=	=	=
queens-fc-10	127526	=	=	=	=
queens-fc-12	3077256	=	=	=	=
queens-gfc-8	902	=	=	=	=
queens-gfc-10	11388	=	=	=	=
queens-gfc-12	248102	=	=	=	=
magic	119	=	=	=	93
zebra	0	=	=	=	=
crypta	3	=	=	=	1
cryptb	4	=	=	=	=
bramsey-12	2961	=	=	2959	2959
bramsey-14	17448	=	=	17444	17444
bramsey-16	10198833	=	=	10198799	10198799
perm-11a	3250	=	=	=	=
perm-11b	10066	=	=	=	=
perm-11c	18427	=	=	=	=
perm-12a	55856	=	=	=	=
perm-12b	59396	=	=	=	=
perm-12c	94174	=	=	=	=
eq-10	99	=	=	=	=
eq-20	60	=	=	=	=

It is also fairly clear that variables which become ground should always be substituted out of all constraints in the store (`ground` vs. `eqn`). Only for a few benchmarks (sched-bridge-good and some perm) does it seem to cause a slight slow-down (presumably due to the extra overhead of checking for substitutions for little or no gain), while it often results in a significant benefit.

Similarly, performing substitution of input UTVPE constraints is almost always worthwhile (utvpe vs. ground). At the extreme, we have the hamil problems, where performing UTVPE substitution results in orders of magnitude improvement. The success on hamil is because performing UTVPE substitutions on disequations increases the strength of the solver so that it finds unsatisfiability much earlier in the search tree. This is reflected in the dramatic reduction in the number of fails for hamil-20 (and, presumably, hamil-30, etc.). This results from sequences of constraints of the form $x_1 \neq x_3$, $x_1 = x_2 + 1$, $x_2 = x_3 - 1$. Using substitution, after reaching the second equation the form of the disequation becomes $0 \neq 0$ and failure is detected immediately. Propagation on the original disequation must wait until either x_1 or x_3 is fixed.

The bramsey problems are the only other ones where search space reductions occur as a result of UTVPE substitutions (albeit minimal ones, and the execution time actually increases ever-so-slightly). It is also worth noting that with the perm problems there was no decrease in the number of fails, but still modest improvements in execution time; this indicates that this optimisation can yield benefit just through more efficient propagation, as well as through its potential for search space reduction.

The final solver option, detecting new UTVPE constraints, does not seem so useful (edetect vs. utvpe). The only benchmarks where it provides an advantage of any significance are the fib-std benchmarks and magic. On the other hand, for the bramsey problems, it suffers a substantial slow-down. The only other benchmarks where new UTVPE constraints occur through substitution are zebra, crypt and eq, and the only benchmarks where there is a reduction in the search space as a result of substituting these new UTVPE constraints are magic and crypta. The lack of advantage of this option is probably due to the detected UTVPE constraints being discovered too late in the computation, so that too little work is saved to offset the cost of substitution. In general, new UTVPE substitutions are probably not worthwhile during an enumerative search, since the chance of them yielding a benefit before they are backtracked is significantly reduced, compared to those performed before the search started, which are substituted once and give benefit throughout the whole search. For the bramsey benchmarks, these factors are compounded by the fact that all the domains are booleans, which means that, in general, less propagation is performed through each constraint (since one bound update makes a variable ground).

The role of clp(FD) in the comparisons is simply to show that our base implementation is reasonable (it is easy to improve a very poor implementation). Our tableau-based solver is competitive with clp(FD). For those problem classes where there was a substantial difference between the two, clp(FD) was faster on three (sched, temp, perm), and our solver was faster on four (hamil, queens, bramsey, eq). In general, it seems that clp(FD) has an advantage when there is a lot of work an intelligent compiler can do, or when the problem is simple. utvpe seems to have an advantage when a significant number of variables appearing in new constraints are ground (thus simplifying the constraints), or where a significant number of the constraints are equations with more than two variables.

5. Conclusion

We have presented theorems that examine when we can safely change the form of a linear constraint in a propagation solver, and guarantee that there is no loss in propagation information. Using these results we have constructed an efficient bounds propagation solver for linear constraints based on a tableau representation and substitution. The solver competes favourably with `clp(FD)`, a stable well-engineered finite domain CLP system. We believe the competitiveness of our solver stems from two things. Firstly, the compact representation of linear constraints used (with efficient update of multiple bounds in two passes over the constraint), rather than the multiple propagation rule representation used in the indexicals scheme (with its redundant computation). We note that most finite domain CLP systems now include specific global constraints for linear sums (for example SICStus Prolog's `scalar_product`) to avoid this inefficiency. Secondly, the ability to easily modify constraints on-the-fly, through variable substitutions.

Although we have presented a solver which handles only linear integer constraints, the same methods can be used even in the presence of nonlinear or other complex constraints which do not support ground or UTVPE substitutions. In such a case, ground substitutions are performed on the linear constraints and nothing further is required. UTVPE substitutions are applied to the linear constraints, and then if the substituted variable occurs in a complex constraint, kept as propagation rules that relate the substituted variable to its replacement expression. In this case we are unable to totally eliminate variables by UTVPE substitutions, but we still restrict the amount of propagation that occurs and get possibly stronger propagation from the linear constraints.

In this paper we have only studied the representation of linear constraints. Nonlinear constraints also have different propagation behaviour depending on representation, and are usually broken into parts (each in a standard form) by finite domain solvers. For nonlinear constraints, the equivalent of Theorem 2 does not hold, and hence an interesting problem to investigate is the most appropriate method for handling nonlinear constraints, in terms of the strength of the resulting domain information.

Notes

1. The number of times each variant solver detected unsatisfiability. We could not instrument `clp(FD)`, but the solver is equivalent in strength to `base`.

References

1. A. Aggoun and N. Beldiceanu. Overview of the CHIP compiler system. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Research*, pages 421–435. MIT Press, 1993.
2. D. Applegate and W. Cook. A computational study of the job-shop scheduling instance. *ORSA Journal on Computing*, 3:149–156, 1991.
3. Philippe Codognet and Daniel Diaz. Compiling constraints in clp(FD). *The Journal of Logic Programming*, 27(3):185–226, 1996.

4. P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions: Mathematical foundations. In *ACM Symposium on Artificial Intelligence and Programming Languages*, 1977.
5. Daniel Diaz and Phillipe Codognet. A minimal extension of the WAM for clp(FD). In David S. Warren, editor, *Proceedings of the 10th International Conference on Logic Programming*, pages 774–790. MIT Press, 1993.
6. Joxan Jaffar, Spiro Michaylov, Peter Stuckey, and Roland Yap. The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, July 1992.
7. S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.
8. J.H.M. Lee, H.F. Leung, and H.W. Won. Extending GENET for non-binary CSP's. In *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence*, pages 338–343. IEEE Computer Society Press, 1995.
9. Jonathan S. Ostroff. *Temporal Logic for Real-Time Systems.* Wiley, 1989.
10. J.F. Puget. A C++ implementation of CLP. Technical report, ILOG SOLVER collected papers, ILOG, 1994.
11. Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. MIT Press, Cambridge, MA, 1989.
12. Pascal Van Hentenryck, Vijay Saraswat, and Yves Deville. Constraint processing in cc(FD). manuscript, 1992.
13. Pascal Van Hentenryck, Vijay Saraswat, and Yves Deville. Design, implementation and evaluation of the constraint language cc(FD). *Journal of Logic Programming*, 37(1–3):139–164, 1998.

Appendix

We present the longest proofs in full in this appendix, to improve the readability of the main body of the text.

THEOREM 2 *Consider the primitive constraint* $c_1 \equiv \sum_{i=1}^n a_i x_i \text{ op } d$, *where* $\text{op} \in \{=, \leq, \neq\}$. *Let* v *be some variable* $v \notin \{x_1, \ldots, x_n\}$. *Then for any subset* $V \subseteq \{1, \ldots, n\}$ *and domain* D *where* $D(v) \supseteq [\sum_{i \in V} \inf_D a_i x_i .. \sum_{i \in V} \sup_D a_i x_i]$, *we have*

$$bsolv(c_1, D) =_{vars(c_1)} bsolv(c_2 \wedge c_3, D)$$

where $c_2 \equiv v + \sum_{i=1, i \notin V}^n a_i x_i \text{ op } d$ *and* $c_3 \equiv v = \sum_{i \in V} a_i x_i$.

Proof: We consider the case for an inequality c_1 first.

We first show that ($\sqsubseteq$) $bsolv(c_1, D) \sqsubseteq_{vars(c_1)} bsolv(c_2 \wedge c_3, D)$. We show that for domains D_1 and E_1 where

$$D_1 \sqsubseteq_{vars(c_1)} E_1 \tag{A.1}$$

$$\inf_{E_1} v \leq \sum_{i \in V} \inf_{D_1} a_i x_i \tag{A.2}$$

$$\sup_{E_1} v \geq \inf \left\{ \sum_{i \in V} \sup_{D_1} a_i x_i, d - \sum_{i \notin V} \inf_{D_1} a_i x_i \right\} \tag{A.3}$$

if $D_2 = bc(c_1, D_1)$ and $E_2 = bc(c_2, E_1) \sqcap bc(c_3, E_1)$ then conditions A.1, A.2 and A.3 continue to hold for D_2 and E_2. If we use $D_1 = E_1 = D$ as the induction base case, the ($\sqsubseteq$) statement then follows from A.1.

For a variable $x_j, 1 \leq j \leq n$ where $a_j > 0$ the possibly new upper bound calculated in $bc(c_1, D_1)$ is given by

$$u_j = \left\lfloor \frac{d - \sum_{i=1, i \neq j}^{n} \inf_{D_1} a_i x_i}{a_j} \right\rfloor$$

The possibly new upper bound calculated by $bc(c_2, E_1)$ if $j \notin V$ is

$$\begin{aligned} u'_j &= \left\lfloor \frac{d - \sum_{i \notin V, i \neq j} \inf_{E_1} a_i x_i - \inf_{E_1} v}{a_j} \right\rfloor \\ &\geq \left\lfloor \frac{d - \sum_{i \notin V, i \neq j} \inf_{D_1} a_i x_i - \sum_{i \in V} \inf_{D_1} a_i x_i}{a_j} \right\rfloor \quad \text{by A.1 and A.2} \\ &= u_j \end{aligned}$$

The possibly new upper bound calculated by $bc(c_3, E_1)$ if $j \in V$ is

$$u'_j = \left\lfloor \frac{\sup_{E_1} v - \sum_{i \in V, i \neq j} \inf_{E_1} a_i x_i}{a_j} \right\rfloor$$

From A.3, either $\sup_{E_1} v \geq d - \sum_{i \notin V} \inf_{D_1} a_i x_i$, in which case

$$\begin{aligned} u'_j &\geq \left\lfloor \frac{d - \sum_{i \notin V} \inf_{D_1} a_i x_i - \sum_{i \in V, i \neq j} \inf_{D_1} a_i x_i}{a_j} \right\rfloor \quad \text{by A.1} \\ &= u_j \end{aligned}$$

or $\sup_{E_1} v \geq \sum_{i \in V} \sup_{D_1} a_i x_i$ so

$$\begin{aligned} u'_j &\geq \left\lfloor \frac{\sum_{i \in V} \sup_{D_1} a_i x_i - \sum_{i \in V, i \neq j} \inf_{D_1} a_i x_i}{a_j} \right\rfloor \quad \text{by A.1} \\ &= \left\lfloor \frac{\sup_{D_1} a_j x_j + \sum_{i \in V, i \neq j} (\sup_{D_1} a_i x_i - \inf_{D_1} a_i x_i)}{a_j} \right\rfloor \\ &\geq \sup_{D_1} x_j \quad \text{since } a_j > 0 \text{ and } \sup_D a_i x_i \geq \inf_D a_i x_i \text{ for any feasible } D \end{aligned}$$

Hence the upper bounds for variables x_j with $a_j > 0$ in D_2 must be no greater than for E_2. The lower bounds for variables x_j where $a_j > 0$ cannot change. Similar manipulations hold for the bounds of variables with $a_j < 0$, and thus $D_2 \sqsubseteq_{vars(c_1)} E_2$ holds (A.1).

It remains to show the conditions A.2 and A.3 continue to hold. They follow directly from the bounds propagation rules for c_2 and c_3 and the fact that $D_2 \sqsubseteq D_1 \sqsubseteq_{vars(c_1)} E_1$.

To show ($\sqsupseteq$) $bsolv(c_1, D) \sqsupseteq_{vars(c_1)} bsolv(c_2 \wedge c_3, D)$, we show that given $D_1 \sqsubseteq_{vars(c_1)} E_1$ and $D_2 = bc(c_3, D_1)$, $D_3 = bc(c_2, D_2)$, $D_4 = bc(c_3, D_3)$ and $E_2 = bc(c_1, E_1)$ then $D_4 \sqsubseteq_{vars(c_1)} E_2$.

By the definition of bounds propagation we have that

$$\inf_{D_2} v \geq \sum_{i \in V} \inf_{D_1} a_i x_i \tag{A.4}$$

$$\sup_{D_3} v \leq d - \sum_{i \notin V} \inf_{D_2} a_i x_i \tag{A.5}$$

Now the upper bounds calculated for E_2 are

$$u_j = \left\lfloor \frac{d - \sum_{i=1, i \neq j}^{n} \inf_{E_1} a_i x_i}{a_j} \right\rfloor$$

We show that the upper bounds for D_4 must be at least as strong. For x_j with $a_j > 0$ there are two cases. If $j \notin V$ then the calculation for D_3 is

$$\begin{aligned}
u'_j &= \left\lfloor \frac{d - \sum_{i \notin V, i \neq j} \inf_{D_2} a_i x_i - \inf_{D_2} v}{a_j} \right\rfloor \\
&\leq \left\lfloor \frac{d - \sum_{i \notin V, i \neq j} \inf_{D_1} a_i x_i - \sum_{i \in V} \inf_{D_1} a_i x_i}{a_j} \right\rfloor \quad \text{by A.4 and } D_2 \sqsubseteq D_1 \\
&\leq \left\lfloor \frac{d - \sum_{i=1, i \neq j}^{n} \inf_{E_1} a_i x_i}{a_j} \right\rfloor \quad \text{since } D_1 \sqsubseteq_{vars(c_1)} E_1 \\
&= u_j
\end{aligned}$$

If $j \in V$ then the calculation for D_4 is

$$\begin{aligned}
u'_j &= \left\lfloor \frac{\sup_{D_3} v - \sum_{i \in V, i \neq j} \inf_{D_3} a_i x_i}{a_j} \right\rfloor \\
&\leq \left\lfloor \frac{d - \sum_{i \notin V} \inf_{D_2} a_i x_i - \sum_{i \in V, i \neq j} \inf_{D_2} a_i x_i}{a_j} \right\rfloor \quad \text{by A.5 and } D_3 \sqsubseteq D_2 \\
&\leq \left\lfloor \frac{d - \sum_{i=1, i \neq j}^{n} \inf_{E_1} a_i x_i}{a_j} \right\rfloor \quad \text{since } D_2 \sqsubseteq D_1 \sqsubseteq_{vars(c_1)} E_1 \\
&= u_j
\end{aligned}$$

The lower bounds of variables x_j with $a_j > 0$ do not change. Similar manipulations hold for the lower bounds for variables with $a_j < 0$, whose upper bounds do not change. Clearly three applications of $biter(c_2 \wedge c_3, D_1)$ is guaranteed to be no weaker than the application of $biter(c_1, E)$. Hence the result holds, by Proposition 1.

The case for equations follows from the above, Lemma 2 and Proposition 1. The case for disequations is simpler since propagation is only possible when all but one original variable x_j is fixed. If $j \notin V$ then v will also be fixed, and a similar propagation will occur (note that v is only fixed when all $x_i, i \in V$ are fixed). If $j \in V$ then by Lemma 4 and Lemma 3 there is a one to one correspondence between bounds of v and x_j and any bounds propagation on v by the disequation will result in a corresponding propagation on x_j. ■

THEOREM 4 *Consider the primitive constraints* $c_1 \equiv \sum_{i=1}^{n} a_i x_i$ op d *where* op $\in \{=, \leq, \neq\}$, *and* $c_2 \equiv b_j x_j + b_k x_k = e, j \neq k, b_j \neq 0, b_k \neq 0$. *Let* $c_3 = subs(c_1, x_j, c_2)$. *Then*

$$dsolv(c_1 \wedge c_2, D) = dsolv(c_3 \wedge c_2, D)$$

for all domains D iff

(a) op $\in \{=, \neq, \leq\}$ *and* $a_j = 0$ *or* $a_k = 0$;

(b) $n = 2$, op $\in \{\leq\}$ *and* $a_j b_j a_k b_k > 0$;

(c) $n = 2$, op $\in \{=\}$ *and* $|a_j b_k / b_j a_k| \neq 1$;

(d) $dsolv(c_1 \wedge c_2, D)$ *gives a false domain for all domains D; or*

(e) $c_3 \leftrightarrow true$.

Proof: Let $D' = dsolv(c_1 \wedge c_2, D)$ and $D'' = dsolv(c_3 \wedge c_2, D)$.

Part A.

To prove the if direction, we examine the cases individually. Theorem 3 gives us that $D' \sqsupseteq D''$, so we need only prove $D' \sqsubseteq D''$ for the cases listed above.

For case (a) suppose $a_j = 0$. Then $c_1 \equiv c_3$, by Lemma 1. Otherwise, $a_k = 0$. Then $x_k \notin vars(c_1)$. We can do a construction similar to that in the proof of Theorem 3, taking $\theta \in D'$ a solution of c_1, and extending this to θ' where $\theta'(x_k) = (e - b_j\theta(x_j))/b_k$. Now θ' is a solution of $c_2 \wedge c_3$ which must be in D''. Thus $D' \sqsubseteq D''$.

Note that for the remaining cases $a_j \neq 0$ and $a_k \neq 0$. This means we can assume, w.l.o.g., that $a_j > 0$, $b_j > 0$ and $b_k > 0$, as follows. If $a_j < 0$ then we can replace x_j everywhere by $-x'_j$ and let $D(x'_j) = \{-d_k | d_k \in D(x_j)\}$. c_2 can then be multiplied by -1 if necessary to make $b_j > 0$. Finally, $b_k < 0$ can be treated analogously to $a_j < 0$.

For case (b) suppose n is 2, op is $\leq$ and $a_j b_j a_k b_k > 0$. Making use of the above assumption, we may also assume that $a_k > 0$. Note that c_3 is of the form $(b_j a_k - a_j b_k) x_k \leq b_j d - a_j e$.

If $(b_j a_k - a_j b_k) = 0$, we have two cases to consider. If $b_j d - a_j e \geq 0$ then $c_3 \leftrightarrow true$, and case (e) applies. Otherwise, $c_3 \leftrightarrow false$, and we need to show that D' is a false domain. Suppose it isn't. Then consider $d_k = \sup_{D'} x_k$, the largest value for x_k in D'. This has a corresponding smallest value (via c_2) for x_j, $d_j = (e - b_k d_k)/b_j$ (if there were a smaller value for x_j, there would be a corresponding larger value for x_k, by Lemma 3, contradicting our assumption that d_k was the largest such value). Now, there must also be a value $d'_j \in D'(x_j)$ such that $a_j d'_j + a_k d_k \leq d$ (c_1). If such a d'_j exists, then d_j must also satisfy the constraint, since d_j is the smallest element of $D'(x_j)$ and $a_j > 0$. Substituting for d_j, we have

$$a_j(e - b_k d_k)/b_j + a_k d_k \leq d$$

$$a_j e + (b_j a_k - a_j b_k) d_k \leq b_j d$$

$$b_j d - a_j e \geq 0$$

However, $b_j d - a_j e < 0$, yielding a contradiction, and hence D' is a false domain.

We now consider when $(b_j a_k - a_j b_k) > 0$. We first show that $(b_j a_k - a_j b_k) d_k \leq b_j d - a_j e$ for all $d_k \in D'(x_k)$. Suppose to the contrary that $d_k = \sup_{D'} x_k$ is such that $(b_j a_k - a_j b_k) d_k > b_j d - a_j e$. Since $d_k \in D'(x_k)$ there must exist $d_j \in D'(x_j)$ such that $a_j d_j + a_k d_k \leq d$ (c_1), and hence $d_j \leq (d - a_k d_k)/a_j$. Also, since $d_j \in D'(x_j)$ there exists $d'_k = (e - b_j d_j)/b_k \in D'(x_k)$ (c_2). Now

$$\begin{aligned} d'_k = \frac{e - b_j d_j}{b_k} &\geq \frac{e - b_j(d - a_k d_k)/a_j}{b_k} \\ &= \frac{a_j e - b_j d + b_j a_k d_k}{a_j b_k} \\ &= \frac{a_j e - b_j d + (b_j a_k - a_j b_k) d_k + a_j b_k d_k}{a_j b_k} \\ &> \frac{a_j e - b_j d + b_j d - a_j e + a_j b_k d_k}{a_j b_k} \quad \text{by assumption} \\ &= d_k \end{aligned}$$

But $d_k = \sup_{D'} x_k$ hence d'_k cannot be in $D'(x_k)$. Contradiction. Hence $(b_j a_k - a_j b_k) d_k \leq b_j d - a_j e$ for all $d_k \in D'(x_k)$. Take any solution θ of c_2 in D'. Then by the above it is also a solution of c_3 and thus $D' \sqsubseteq D''$.

Finally, if $(b_j a_k - a_j b_k) < 0$, we can use the same approach to show $(b_j a_k - a_j b_k) d_k \leq b_j d - a_j e$. Let $d_k = \inf_{D'} x_k$, $d_j = (e - b_k d_k)/b_j$, $d'_k \leq (d - a_j d_j)/a_k$. If $(b_j a_k - a_j b_k) d_k > b_j d - a_j e$ then we can show $d'_k < d_k$.

For case (c), we have $n = 2$, $op \in \{=\}$ and $|a_j b_k / b_j a_k| \neq 1$. If there is a solution, the constraint on the coefficients guarantees it to be unique. If there is no solution, then case (d) applies and we need not consider it here. Let the unique solution be $x_j = d_j$, $x_k = d_k$. Note that we have

$$(b_j a_k - a_j b_k) d_k = b_j d - a_j e \tag{A.6}$$

We show that $D'(x_j) = d_j$ and $D'(x_k) = d_k$. There are two cases we consider.

Case $|a_j b_k / b_j a_k| > 1$:

Let $d'_k = \sup_{D'} x_k$ and suppose $d'_k \neq d_k$. Then by Lemma 3 and c_2, we have $d'_j = (e - b_k d'_k)/b_j \in D'(x_j)$. Applying Lemma 3 again with c_1, we also have $d''_k = (d - a_j d'_j)/a_k \in D'(x_k)$.

$$\begin{aligned} d''_k &= \frac{b_j d - a_j e + a_j b_k d'_k}{b_j a_k} \\ &= \frac{b_j a_k d_k - a_j b_k d_k + a_j b_k d'_k}{b_j a_k} \quad \text{by A.6} \\ &= d_k + \frac{a_j b_k}{b_j a_k}(d'_k - d_k) \end{aligned}$$

Iterating the above process again yields

$$d'''_k = d_k + \left(\frac{a_j b_k}{b_j a_k}\right)^2 (d'_k - d_k) \in D'(x_k)$$

Clearly, if $|a_j b_k / b_j a_k| > 1$ and $d'_k \neq d_k$ then $d'''_k > d'_k$, contradicting our assumption that there exists a $d'_k \neq d_k$ which is the largest element of $D'(x_k)$. Hence d_k is the largest element of $D'(x_k)$. A similar argument shows that d_k is also the smallest element of $D'(x_k)$. Hence $D'(x_k) = d_k$. That $D'(x_j) = d_j$ follows immediately.

Case $|a_j b_k / b_j a_k| < 1$:

This case simply uses the constraints in the reverse order. Here, let $d'_j = (d - a_k d'_k)/a_j \in D'(x_j)$ (by c_1) and $d''_k = (e - b_j d'_j)/b_k \in D'(x_k)$ (by c_2). Then

$$\begin{aligned} d''_k &= \frac{a_j e - b_j d + b_j a_k d'_k}{a_j b_k} \\ &= \frac{a_j b_k d_k - b_j a_k d_k + b_j a_k d'_k}{a_j b_k} \quad \text{by A.6} \\ &= d_k + \frac{b_j a_k}{a_j b_k}(d'_k - d_k) \end{aligned}$$

$|b_j a_k / a_j b_k| > 1$ and the result follows as above.

Cases (d) and (e) are straightforward. For case (d), if $\mathit{dsolv}(c_1 \wedge c_2, D)$ gives a false domain for all domains D then the result holds trivially. For case (e), if $c_3 \leftrightarrow \mathit{true}$ then $\mathit{dsolv}(c_2 \wedge c_3, D) = \mathit{dsolv}(c_2, D) \sqsupseteq \mathit{dsolv}(c_1 \wedge c_2, D)$.

Part B.

For the only if direction we construct a domain D for any c_1, c_2 for the cases where none of the above conditions hold, such that $\mathit{dsolv}(c_1 \wedge c_2, D) \sqsupset \mathit{dsolv}(c_3 \wedge c_2, D)$.

Suppose $c_1 \wedge c_2 \leftrightarrow c_2 \wedge c_3$ has no solution. Then $D'' = \mathit{dsolv}(c_2 \wedge c_3, D)$ for any D is a false domain because any solution $\theta \in D''$ of c_3 can be extended to a solution of $c_2 \wedge c_3$ by choosing the appropriate value of x_j and we have a contradiction. Since $\mathit{dsolv}(c_2 \wedge c_3, D)$ is a false domain for any D the result holds by choosing a domain D such that $\mathit{dsolv}(c_1 \wedge c_2, D)$ is not a false domain. Such a domain exists because case (d) does not apply.

Hence we can assume for the remainder of the proof that there is an integer solution for $c_1 \wedge c_2 \leftrightarrow c_2 \wedge c_3$. We can also assume $a_j \neq 0$ and $a_k \neq 0$, or case (a) applies.

We assume that $g = \gcd_{i=1}^n(a_i) = 1$ (otherwise divide c_1 by g: the constraints are equivalent by Lemma 1). Similarly, we assume $\gcd(b_j, b_k) = 1$. Finally, we assume, as above, that $a_j > 0$, $b_j > 0$, $b_k > 0$.

We first tackle the case where op is $\leq$. If $n = 2$, we may assume $a_k < 0$ (so that case (b) does not apply), which means $b_j a_k - a_j b_k < 0$. Then

$$c_3 \;\equiv\; x_k \geq \frac{b_j d - a_j e}{b_j a_k - a_j b_k}$$

Let $x_k = d_k$ be the smallest solution of c_3 which also appears in a solution of c_2. Let $d_j = (e - b_k d_k)/b_j$ be the corresponding (via c_2) value of x_j. Let $D(x_j) = \{d_j + m b_k \mid m \in [a_j b_j a_k b_k..1]\}$, $D(x_k) = \{d_k - m b_j \mid m \in [a_j b_j a_k b_k..1]\}$. We claim $D' = \mathit{dsolv}(c_1 \wedge c_2, D) = D$, and $D'' = \mathit{dsolv}(c_3 \wedge c_2, D)$ excludes the elements corresponding to $m = 1$.

Consider $d'_j = d_j + m b_k \in D(x_j)$. $\theta = \{x_j \mapsto d_j + m b_k, x_k \mapsto d_k - m b_j\}$ is a solution of c_2. Also, $\theta' = \{x_j \mapsto d_j + m b_k, x_k \mapsto d_k - (a_j b_j a_k b_k) b_j\}$ is a solution of c_1. ((d_j, d_k) is a solution, and $ma_jb_k - (a_jb_ja_kb_k)b_ja_k = a_jb_k(m - b_j^2a_k^2) \leq 0$). Hence $d'_j \in D'(x_j)$. A similar construction shows $d_k - mb_j \in D'(x_k)$.

However, $d_k - b_j \notin D''(x_k)$, since $d_k - b_j < d_k$, and d_k is by construction the smallest value of x_k appearing in solutions of c_2 and c_3. Hence $D'' \sqsubset D'$.

We can use the same proof for $n \geq 3$ where $a_k < 0$ by simply choosing a domain which fixes all the other variables.

That just leaves the $n \geq 3$, $a_k > 0$ case for op $\in \{\leq\}$. Let $g' = \gcd_{i\neq j, i\neq k}(a_i)$. Let $v = \frac{1}{g'}\sum_{i=1,i\neq j,i\neq k}^{n} a_ix_i$, $v \notin \{x_1, \ldots, x_n\}$. Let $(x_j, x_k) = (d_j, d_k)$ be any solution of c_2. Then $(d_j + mb_k, d_k - mb_j)$ for integral m are all the solutions of c_2. Let $(m, v) = (m_0, v_0)$ be any solution of $a_j(d_j + mb_k) + a_k(d_k - mb_j) + g'v = d'$ which maximises d' subject to $d' \leq d$. Note that for any integer m', $(m_0 + m'g', v_0 - m'a_jb_k + m'b_ja_k)$ is also a solution which maximises d'. Note also that $d - d' < |g'|$.

Choose some $l \neq j, l \neq k$ such that $a_l \neq 0$. Note g' divides a_l, so $|g'| \leq |a_l|$ and hence $d - d' < |a_l|$. Let $d'_j = d_j + m_0b_k$, $d'_k = d_k - m_0b_j$, $D(x_j) = \{d'_j, d'_j + a_lb_k\}$ and $D(x_k) = \{d'_k, d'_k - a_lb_j\}$. Let θ be a solution to $\sum_{i=1,i\neq j,i\neq k}^{n} a_ix_i = g'v_0$. Let $D(x_i) = \{\theta(x_i)\}$ for all $i \neq j, i \neq k, i \neq l$, and let $D(x_l) \supseteq \{\theta(x_l), \theta(x_l) + b_ja_k, \theta(x_l) - a_jb_k, \theta(x_l) - a_jb_k + b_ja_k\}$.

Suppose $a_l > 0$ (the other case is similar). Then $dc(c_1, D)$ yields

$$
\begin{aligned}
x_l &\leq \left\lfloor \frac{d - \sum_{i=1,i\neq l}^{n} \inf_D a_ix_i}{a_l} \right\rfloor \\
&= \left\lfloor \frac{d - \sum_{i=1,i\neq j,i\neq k,i\neq l}^{n} a_i\theta(x_i) - \inf_D a_jx_j - \inf_D a_kx_k}{a_l} \right\rfloor \\
&= \left\lfloor \frac{d - g'v_0 + a_l\theta(x_l) - \inf_D a_jx_j - \inf_D a_kx_k}{a_l} \right\rfloor \\
&= \left\lfloor \frac{a_l\theta(x_l) + d - d' + a_jd'_j + a_kd'_k - \inf_D a_jx_j - \inf_D a_kx_k}{a_l} \right\rfloor \\
&= \left\lfloor \frac{a_l\theta(x_l) + d - d' + a_l \sup\{0, b_ja_k, -a_jb_k, b_ja_k - a_jb_k\}}{a_l} \right\rfloor \\
&= \theta(x_l) + \left\lfloor \frac{d - d'}{a_l} \right\rfloor + \sup\{0, b_ja_k, -a_jb_k, b_ja_k - a_jb_k\} \\
&= \theta(x_l) + \sup\{0, b_ja_k, -a_jb_k, b_ja_k - a_jb_k\}.
\end{aligned}
$$

Similarly, $dc(c_3, D)$ yields

$$
\begin{aligned}
x_l &\leq \left\lfloor \frac{b_jd - a_je - \inf_D(b_ja_k - a_jb_k)x_k - b_j\sum_{i=1,i\neq j,i\neq k,i\neq l}^{n} \inf_D a_ix_i}{b_ja_l} \right\rfloor \\
&= \left\lfloor \frac{b_jd - a_je - (b_ja_k - a_jb_k)d'_k + b_ja_l \sup\{0, b_ja_k - a_jb_k\} - b_jg'v_0 + b_ja_l\theta(x_l)}{b_ja_l} \right\rfloor \\
&= \left\lfloor \frac{b_jd - a_jb_jd'_j - b_ja_kd'_k + b_ja_l \sup\{0, b_ja_k - a_jb_k\} - b_j(d' - a_jd'_j - a_kd'_k) + b_ja_l\theta(x_l)}{b_ja_l} \right\rfloor
\end{aligned}
$$

$$
\begin{aligned}
&= \left\lfloor \frac{d + a_l \sup\{0, b_j a_k - a_j b_k\} - d' + a_l \theta(x_l)}{a_l} \right\rfloor \\
&= \left\lfloor \frac{d - d' + a_l \theta(x_l) + a_l \sup\{0, b_j a_k - a_j b_k\}}{a_l} \right\rfloor \\
&= \theta(x_l) + \sup\{0, b_j a_k - a_j b_k\}.
\end{aligned}
$$

(Note that in each case further iterations of dc do not yield any new bounds.) Since $a_j b_j a_k b_k > 0$, the latter bound is strictly tighter, and in particular excludes $\theta(x_l) + b_j a_k$. Thus $D' \sqsupset D''$.

We now consider the case where op is $=$.

If $n = 2$ and $b_j a_k / a_j b_k = 1$ then $b_j a_k = a_j b_k$, and (e) applies (since we have already covered the case where there is no solution).

If $n = 2$ and $b_j a_k / a_j b_k = -1$, then $c_3 \equiv 2 b_j a_k x_k = b_j d - a_j e$. As the result is already proved for the case of no solutions, we may assume a unique solution $x_k = d_k$, $x_j = d_j$. Consider the domain D such that

$$
\begin{aligned}
D(x_j) &= \{d_j - b_k, d_j, d_j + b_k\} \\
D(x_k) &= \{d_k - b_j, d_k, d_k + b_j\}
\end{aligned}
$$

Clearly $D'' = dsolv(c_3 \wedge c_2, D)$ includes just the unique solution; however, $D' = dsolv(c_1 \wedge c_2, D) = D$. To see this, note that each element of each domain has a support value in the other domain for both c_1 and c_2. For c_1, with $m \in \{-1, 0, 1\}$:

$$
\begin{aligned}
& a_j(d_j + m b_k) + a_k(d_k + m b_j) \\
= \;& a_j d_j + a_k d_k + m(a_j b_k + b_j a_k) \\
= \;& d
\end{aligned}
$$

For c_2:

$$
\begin{aligned}
& b_j(d_j + m b_k) + b_k(d_k - m b_j) \\
= \;& b_j d_j + b_k d_k + m(b_j b_k - b_j b_k) \\
= \;& e
\end{aligned}
$$

For op $\in \{=\}$, this just leaves the case where $n \geq 3$, with $a_j \neq 0$ and $a_k \neq 0$. The same approach is used as for op $\in \{\leq\}$, but with $d' = d$. In this case, $D'(x_l) = \{\theta(x_l), \theta(x_l) + b_j a_k, \theta(x_l) - a_j b_k, \theta(x_l) - a_j b_k + b_j a_k\}$ and $D''(x_l) = \{\theta(x_l), \theta(x_l) - a_j b_k + b_j a_k\}$. Thus $D' \sqsupset D''$ since $a_j \neq 0$ and $a_k \neq 0$.

The final case is when op is $\neq$. Let $c_1^=$ be c_1 with the $\neq$ replaced by $=$. Either there exists a solution θ to $c_1^= \wedge c_2$ or $c_3 \leftrightarrow true$. Let $\theta' \neq \theta$ be another solution of c_2. Define $D(x_j) = \{\theta(x_j), \theta'(x_j)\}$, $D(x_k) = \{\theta(x_k), \theta'(x_k)\}$, and $D(x_i) = \{\theta(x_i)\}$ for any remaining variables. Then $dsolv(c_1 \wedge c_2, D) = D$ but $dsolv(c_2 \wedge c_3, D)(x_k)$ does not include $\theta(x_k)$. ■

THEOREM 5 *Consider the primitive constraints* $c_1 \equiv \sum_{i=1}^{n} a_i x_i$ op d *where* op $\in \{=, \leq, \neq\}$, *and* $c_2 \equiv b_j x_j + b_k x_k = e$, $j \neq k$, $b_j \neq 0$, $b_k \neq 0$. *Let* $c_3 = subs(c_1, x_j, c_2)$. *Then for all domains* D *we have*

$$bsolv(c_1 \wedge c_2, D) \sqsupseteq bsolv(c_3 \wedge c_2, D)$$

Proof: The case for $a_j = 0$ follows from Lemma 1, so we assume $a_j \neq 0$. W.l.o.g. we can assume not only that $b_j > 0$ but also that $b_k > 0$ and $a_j > 0$, as before.

We prove the case where c_1 is an inequality. By Proposition 1 we have that $bsolv(c_1 \wedge c_2, D) = \text{gfp}(\lambda d. bc(c_1, bsolv(c_2, d)))(D)$ and similarly for $c_3 \wedge c_2$. Suppose $D_1 \sqsupseteq E_1$ and

$$
\begin{aligned}
D_2 &= bsolv(c_2, D_1) \\
D_3 &= bc(c_1, D_2) \\
E_2 &= bsolv(c_2, E_1) \\
E_3 &= bc(c_3, E_2) \\
E_4 &= bc(c_2, E_3)
\end{aligned}
$$

We prove that $D_3 \sqsupseteq E_4$. Clearly, $D_2 \sqsupseteq E_2$ and $D_3 \sqsupseteq E_3 \Rightarrow D_3 \sqsupseteq E_4$, so that in the following we will often just show that $D_2 \sqsupseteq E_2 \Rightarrow D_3 \sqsupseteq E_3$. Note also that we may assume that E_4 (and hence E_3, etc.) is a satisfiable domain, or the result holds trivially.

By Lemma 3 we have that

$$b_j \inf_{D_2} x_j + b_k \sup_{D_2} x_k = e \tag{A.7}$$

$$b_j \sup_{D_2} x_j + b_k \inf_{D_2} x_k = e \tag{A.8}$$

$$b_j \inf_{E_2} x_j + b_k \sup_{E_2} x_k = e \tag{A.9}$$

$$b_j \sup_{E_2} x_j + b_k \inf_{E_2} x_k = e \tag{A.10}$$

These identities will be used often below.

We focus on the case where op $\in \{\leq\}$. We first consider the calculation of upper bounds for x_l where $a_l > 0$, $l \notin \{j, k\}$, in the evaluation of $D_3 = bc(c_1, D_2)$ and $E_3 = bc(c_3, E_2)$.

The (possibly new) upper bound calculated for D_3 is

$$u_l = \left\lfloor \frac{d - \sum_{i=1, i \neq l}^{n} \inf_{D_2} a_i x_i}{a_l} \right\rfloor$$

while the (possibly new) upper bound for E_3 is

$$u'_l = \left\lfloor \frac{b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k,l\}}^{n} \inf_{E_2} a_i x_i - \inf_{E_2} (b_j a_k - a_j b_k) x_k}{b_j a_l} \right\rfloor$$

To show that $u'_l \leq u_l$ we need to show that

$$-b_j \sum_{i=1, i \neq l}^{n} \inf_{D_2} a_i x_i \geq -a_j e - b_j \sum_{i=1, i \notin \{j,k,l\}}^{n} \inf_{E_2} a_i x_i - \inf_{E_2} (b_j a_k - a_j b_k) x_k$$

Now

$$\begin{aligned}
& a_j e + b_j \sum_{i=1, i \notin \{j,k,l\}}^{n} \inf_{E_2} a_i x_i + \inf_{E_2}(b_j a_k - a_j b_k) x_k - b_j \sum_{i=1, i \neq l}^{n} \inf_{D_2} a_i x_i \\
\geq\ & a_j e + b_j \sum_{i=1, i \notin \{j,k,l\}}^{n} \inf_{E_2} a_i x_i + \inf_{E_2}(b_j a_k - a_j b_k) x_k - b_j \sum_{i=1, i \neq l}^{n} \inf_{E_2} a_i x_i \\
& \quad \text{since } D_2 \sqsupseteq E_2 \\
=\ & a_j e + \inf_{E_2}(b_j a_k - a_j b_k) x_k - b_j \inf_{E_2} a_j x_j - b_j \inf_{E_2} a_k x_k \\
=\ & a_j e + \inf_{E_2}(b_j a_k - a_j b_k) x_k - a_j b_j \inf_{E_2} x_j - b_j \inf_{E_2} a_k x_k \\
=\ & a_j e + \inf_{E_2}(b_j a_k - a_j b_k) x_k - a_j (e - b_k \sup_{E_2} x_k) - b_j \inf_{E_2} a_k x_k \quad \text{by (A.9)} \\
=\ & \inf_{E_2}(b_j a_k - a_j b_k) x_k + a_j b_k \sup_{E_2} x_k - b_j \inf_{E_2} a_k x_k \\
=\ & \inf_{E_2}(b_j a_k - a_j b_k) x_k - \inf_{E_2}(-a_j b_k x_k) - \inf_{E_2} b_j a_k x_k \\
\geq\ & 0 \quad \text{since } \inf(a+b)x \geq \inf ax + \inf bx
\end{aligned}$$

Hence $u_l' \leq u_l$, and $D_3(x_l) \sqsupseteq E_3(x_l)$ (for $a_l > 0$).

If $a_l < 0$, the converse condition, swapping sup for inf, needs to be proved for the lower bound of x_l. The result follows analogously.

We now need to prove the result for x_j and x_k.

For x_k, we start with the case where $b_j a_k - a_j b_k = 0$. We show that c_1 cannot impose a new bound on x_k, so $D_3(x_k) = D_2(x_k)$ and hence $D_3(x_k) \sqsupseteq E_3(x_k)$.

Consider the case where $a_k > 0$ (the $a_k < 0$ case follows analogously). The upper bound on x_k given by c_1 is

$$u_k = \left\lfloor \frac{d - \sum_{i=1, i \neq k}^{n} \inf_{D_2} a_i x_i}{a_k} \right\rfloor$$

Since we may assume that propagating through c_3 does not result in an unsatisfiable domain, we have that

$$\begin{aligned}
& b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i \ \leq\ b_j d - a_j e \\
\Rightarrow\ & b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i \ \leq\ b_j d - a_j e \quad \text{since } D_2 \sqsupseteq E_2
\end{aligned} \tag{A.11}$$

Now

$$\begin{aligned}
u_k\ &=\ \left\lfloor \frac{d - \sum_{i=1, i \neq k}^{n} \inf_{D_2} a_i x_i}{a_k} \right\rfloor \\
&=\ \left\lfloor \frac{b_j d - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i - b_j \inf_{D_2} a_j x_j}{b_j a_k} \right\rfloor
\end{aligned}$$

$$
\begin{aligned}
&= \left\lfloor \frac{b_j d - b_j \sum_{i=1, i\notin\{j,k\}}^{n} \inf_{D_2} a_i x_i - a_j(e - \sup_{D_2} b_k x_k)}{b_j a_k} \right\rfloor \quad \text{by (A.7)} \\
&= \left\lfloor \frac{b_j d - a_j e - b_j \sum_{i=1, i\notin\{j,k\}}^{n} \inf_{D_2} a_i x_i + a_j \sup_{D_2} b_k x_k}{b_j a_k} \right\rfloor \\
&\geq \left\lfloor \frac{a_j b_k \sup_{D_2} x_k}{b_j a_k} \right\rfloor \quad \text{by (A.11)} \\
&= \sup_{D_2} x_k \quad \text{since } b_j a_k - a_j b_k = 0
\end{aligned}
$$

Thus c_1 cannot impose a new bound on x_k and the result follows.

We next consider the case where $b_j a_k - a_j b_k > 0$, so that $b_j a_k > a_j b_k$ and $a_k > 0$. Then the (possibly new) upper bound on x_k given by c_1 is

$$u_k = \left\lfloor \frac{d - \sum_{i=1, i\neq k}^{n} \inf_{D_2} a_i x_i}{a_k} \right\rfloor$$

We will show that this is no smaller than the upper bound on x_k in E_3:

$$u'_k = \sup_{E_3} x_k = \min\left(\sup_{E_2} x_k, \left\lfloor \frac{b_j d - a_j e - b_j \sum_{i=1, i\notin\{j,k\}}^{n} \inf_{E_2} a_i x_i}{b_j a_k - a_j b_k} \right\rfloor\right)$$

Now

$$
\begin{aligned}
u'_k &\leq \frac{b_j d - a_j e - b_j \sum_{i=1, i\notin\{j,k\}}^{n} \inf_{E_2} a_i x_i}{b_j a_k - a_j b_k} \\
\Rightarrow \quad (b_j a_k - a_j b_k) u'_k &\leq b_j d - a_j e - b_j \sum_{i=1, i\notin\{j,k\}}^{n} \inf_{E_2} a_i x_i \\
\Rightarrow \quad b_j \sum_{i=1, i\notin\{j,k\}}^{n} \inf_{E_2} a_i x_i &\leq b_j d - a_j e - (b_j a_k - a_j b_k) u'_k \\
\Rightarrow \quad b_j \sum_{i=1, i\notin\{j,k\}}^{n} \inf_{D_2} a_i x_i &\leq b_j d - a_j e - (b_j a_k - a_j b_k) u'_k \qquad \text{(A.12)} \\
&\qquad \text{since } D_2 \sqsupseteq E_2
\end{aligned}
$$

Then

$$
\begin{aligned}
u_k &= \left\lfloor \frac{d - \sum_{i=1, i\notin\{j,k\}}^{n} \inf_{D_2} a_i x_i - \inf_{D_2} a_j x_j}{a_k} \right\rfloor \\
&= \left\lfloor \frac{b_j d - b_j \sum_{i=1, i\notin\{j,k\}}^{n} \inf_{D_2} a_i x_i - b_j \inf_{D_2} a_j x_j}{b_j a_k} \right\rfloor \\
&\geq \left\lfloor \frac{b_j d - (b_j d - a_j e - (b_j a_k - a_j b_k) u'_k) - b_j a_j \inf_{D_2} x_j}{b_j a_k} \right\rfloor \quad \text{by (A.12)}
\end{aligned}
$$

$$\begin{aligned}
&= \left\lfloor \frac{a_j e + (b_j a_k - a_j b_k) u'_k - a_j (e - \sup_{D_2} b_k x_k)}{b_j a_k} \right\rfloor \quad \text{by (A.7)} \\
&= \left\lfloor u'_k + \frac{a_j b_k (\sup_{D_2} x_k - u'_k)}{b_j a_k} \right\rfloor \\
&\geq u'_k \quad \text{since } u'_k \leq \sup_{E_2} x_k \leq \sup_{D_2} x_k
\end{aligned}$$

as required.

For the case where $b_j a_k - a_j b_k < 0$ then $b_j a_k < a_j b_k$ and we need to consider two cases. When $a_k > 0$ we show that either the new upper bound u_k from c_1 is no stronger than the previous one (i.e. $\sup_{D_3} x_k = \sup_{D_2} x_k \geq \sup_{E_2} x_k$), or the new lower bound l'_k from c_3 gives a false domain ($l'_k > \sup_{E_2} x_k$).

The (possibly new) lower bound on x_k given by c_3 is

$$l'_k = \left\lceil \frac{b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i}{b_j a_k - a_j b_k} \right\rceil$$

Since we may assume $l'_k \leq \sup_{E_2} x_k$ we have

$$\begin{aligned}
\sup_{D_2} x_k \geq \sup_{E_2} x_k \ &\geq \ \frac{b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i}{b_j a_k - a_j b_k} \\
\Rightarrow \ (b_j a_k - a_j b_k) \sup_{D_2} x_k \ &\leq \ b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i \\
\Rightarrow \ b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i \ &\leq \ b_j d - a_j e - (b_j a_k - a_j b_k) \sup_{D_2} x_k \\
\Rightarrow \ b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i \ &\leq \ b_j d - a_j e - (b_j a_k - a_j b_k) \sup_{D_2} x_k \qquad \text{(A.13)} \\
&\text{since } D_2 \sqsupseteq E_2
\end{aligned}$$

Now the (possibly new) upper bound on x_k given by c_1 is

$$\begin{aligned}
u_k &= \left\lfloor \frac{d - \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i - \inf_{D_2} a_j x_j}{a_k} \right\rfloor \\
&= \left\lfloor \frac{b_j d - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i - b_j \inf_{D_2} a_j x_j}{b_j a_k} \right\rfloor \\
&\geq \left\lfloor \frac{b_j d - (b_j d - a_j e - (b_j a_k - a_j b_k) \sup_{D_2} x_k) - b_j a_j \inf_{D_2} x_j}{b_j a_k} \right\rfloor \quad \text{by (A.13)} \\
&= \left\lfloor \frac{a_j e + (b_j a_k - a_j b_k) \sup_{D_2} x_k - a_j (e - b_k \sup_{D_2} x_k)}{b_j a_k} \right\rfloor \quad \text{by (A.7)} \\
&= \left\lfloor \frac{b_j a_k \sup_{D_2} x_k}{b_j a_k} \right\rfloor \\
&= \sup_{D_2} x_k
\end{aligned}$$

Thus c_1 does not impose a new upper bound on x_k.

The final case is where $b_j a_k - a_j b_k < 0$ and $a_k < 0$. We prove that the (possibly new) lower bound on x_k given by c_1

$$l_k = \left\lceil \frac{d - \sum_{i=1, i \notin \{j,k\}}^n \inf_{D_2} a_i x_i - \inf_{D_2} a_j x_j}{a_k} \right\rceil$$

is no greater than the (possibly new) lower bound on x_k given by c_3

$$l'_k = \left\lceil \frac{b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^n \inf_{E_2} a_i x_i}{b_j a_k - a_j b_k} \right\rceil$$

We have that

$$\begin{aligned}
l'_k &\geq \frac{b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^n \inf_{E_2} a_i x_i}{b_j a_k - a_j b_k} \\
\Rightarrow \quad (b_j a_k - a_j b_k) l'_k &\leq b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^n \inf_{E_2} a_i x_i \\
\Rightarrow \quad b_j \sum_{i=1, i \notin \{j,k\}}^n \inf_{E_2} a_i x_i &\leq b_j d - a_j e - (b_j a_k - a_j b_k) l'_k \\
\Rightarrow \quad b_j \sum_{i=1, i \notin \{j,k\}}^n \inf_{D_2} a_i x_i &\leq b_j d - a_j e - (b_j a_k - a_j b_k) l'_k \qquad \text{(A.14)} \\
& \qquad \text{since } D_2 \sqsupseteq E_2
\end{aligned}$$

Now

$$\begin{aligned}
l_k &= \left\lceil \frac{d - \sum_{i=1, i \notin \{j,k\}}^n \inf_{D_2} a_i x_i - \inf_{D_2} a_j x_j}{a_k} \right\rceil \\
&= \left\lceil \frac{b_j d - b_j \sum_{i=1, i \notin \{j,k\}}^n \inf_{D_2} a_i x_i - b_j \inf_{D_2} a_j x_j}{b_j a_k} \right\rceil \\
&\leq \left\lceil \frac{b_j d - (b_j d - a_j e - (b_j a_k - a_j b_k) l'_k) - b_j \inf_{D_2} a_j x_j}{b_j a_k} \right\rceil \quad \text{by (A.14), } a_k < 0 \\
&= \left\lceil \frac{a_j e + (b_j a_k - a_j b_k) l'_k - a_j b_j \inf_{D_2} x_j}{b_j a_k} \right\rceil \\
&= \left\lceil \frac{a_j e + (b_j a_k - a_j b_k) l'_k - a_j (e - \sup_{D_2} b_k x_k)}{b_j a_k} \right\rceil \quad \text{by (A.7)} \\
&= \left\lceil l'_k + \frac{a_j b_k (\sup_{D_2} x_k - l'_k)}{b_j a_k} \right\rceil \\
&\leq l'_k \quad \text{since } \sup_{D_2} x_k \geq \sup_{E_2} x_k \geq l'_k \text{ and } a_k < 0
\end{aligned}$$

Thus we have that $D_3(x_k) \sqsupseteq E_3(x_k)$.

Now it remains to prove the cases for x_j. The $b_j a_k - a_j b_k = 0$ case follows from the same reasoning used for x_k, with j and k exchanged. So suppose $b_j a_k - a_j b_k > 0$ then $a_k > 0$. We show that either the upper bound u_j from c_1 is larger than the previous upper bound $\sup_{D_2} x_j$ or the upper bound u'_k for x_k from c_3 is smaller than the previous lower bound $\inf_{E_2} x_k$ (i.e. E_3 is a false domain).

$$u_j = \left\lfloor \frac{d - \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i - \inf_{D_2} a_k x_k}{a_j} \right\rfloor$$

And the new upper bound for x_k from c_3 is

$$u'_k = \left\lfloor \frac{b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i}{b_j a_k - a_j b_k} \right\rfloor$$

Assume $u'_k \geq \inf_{E_2} x_k \geq \inf_{D_2} x_k$. Then

$$\begin{aligned}
\inf_{D_2} x_k \leq u'_k &\leq \frac{b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i}{b_j a_k - a_j b_k} \\
(b_j a_k - a_j b_k) \inf_{D_2} x_k &\leq b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i \\
b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i &\leq b_j d - a_j e - (b_j a_k - a_j b_k) \inf_{D_2} x_k \\
b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i &\leq b_j d - a_j e - (b_j a_k - a_j b_k) \inf_{D_2} x_k \qquad \text{(A.15)} \\
&\text{since } D_2 \sqsupseteq E_2
\end{aligned}$$

Now we have that

$$\begin{aligned}
u_j &= \left\lfloor \frac{b_j d - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i - b_j \inf_{D_2} a_k x_k}{b_j a_j} \right\rfloor \\
&\geq \left\lfloor \frac{b_j d - (b_j d - a_j e - (b_j a_k - a_j b_k) \inf_{D_2} x_k) - b_j \inf_{D_2} a_k x_k}{b_j a_j} \right\rfloor \quad \text{by (A.15)} \\
&= \left\lfloor \frac{a_j e + (b_j a_k - a_j b_k) \inf_{D_2} x_k - b_j a_k \inf_{D_2} x_k}{b_j a_j} \right\rfloor \\
&= \left\lfloor \frac{a_j e - a_j b_k \inf_{D_2} x_k}{b_j a_j} \right\rfloor \\
&= \left\lfloor \frac{e - (e - \sup_{D_2} b_j x_j)}{b_j} \right\rfloor \quad \text{by (A.8)} \\
&= \sup_{D_2} x_j
\end{aligned}$$

The other case is $b_j a_k - a_j b_k < 0$. We show that the upper bound u_j for x_j from c_1 is no smaller than the upper bound u''_j obtained by propagating the lower bound

$l'_k = \inf_{E_3} x_k$ through c_2 (i.e. $\sup_{D_3} x_j \geq \sup_{E_4} x_j$).

$$\begin{aligned}
u_j &= \left\lfloor \frac{d - \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i - \inf_{D_2} a_k x_k}{a_j} \right\rfloor \\
l'_k &= \sup \left\{ \left\lceil \frac{b_j d - a_j e - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{E_2} a_i x_i}{b_j a_k - a_j b_k} \right\rceil, \inf_{E_2} x_k \right\} \\
u''_j &= \left\lfloor \frac{e - b_k l'_k}{b_j} \right\rfloor
\end{aligned}$$

Now $u''_j \leq \frac{e - b_k l'_k}{b_j}$ hence

$$l'_k \leq \frac{e - b_j u''_j}{b_k} \tag{A.16}$$

Note also that (A.14) still holds under these conditions:

$$b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i \;\leq\; b_j d - a_j e - (b_j a_k - a_j b_k) l'_k$$

Now

$$\begin{aligned}
u_j &= \left\lfloor \frac{b_j d - b_j \sum_{i=1, i \notin \{j,k\}}^{n} \inf_{D_2} a_i x_i - b_j \inf_{D_2} a_k x_k}{b_j a_j} \right\rfloor \\
&\geq \left\lfloor \frac{b_j d - (b_j d - a_j e - (b_j a_k - a_j b_k) l'_k) - b_j \inf_{D_2} a_k x_k}{b_j a_j} \right\rfloor \quad \text{by (A.14)} \\
&= \left\lfloor \frac{a_j e + b_j (a_k l'_k - \inf_{D_2} a_k x_k) - a_j b_k l'_k}{b_j a_j} \right\rfloor \\
&\geq \left\lfloor \frac{a_j e + b_j (a_k l'_k - \inf_{D_2} a_k x_k) - a_j (e - b_j u''_j)}{b_j a_j} \right\rfloor \quad \text{by (A.16)} \\
&= u''_j + \left\lfloor \frac{a_k l'_k - \inf_{D_2} a_k x_k}{a_j} \right\rfloor \\
&\geq u''_j \quad \text{since (when } a_k > 0) \ l'_k \geq \inf_{E_2} x_k \geq \inf_{D_2} x_k \\
&\qquad \text{and } (a_k < 0) \text{ either } l'_k \leq \sup_{D_2} x_k \text{ or the result is a false domain}
\end{aligned}$$

Thus we have $D_3(x_j) \sqsupseteq E_4(x_j)$, which completes the proof of $D_3 \sqsupseteq E_4$, and hence we have that

$$bsolv(c_1 \wedge c_2, D) \sqsupseteq bsolv(c_3 \wedge c_2, D)$$

for op $\in \{\leq\}$.

We can prove the equation case using the cases for upper bounds and lower bounds together. The case for disequations is different but simpler. ∎