When Do Bounds and Domain Propagation Lead to the Same Search Space

CHRISTIAN SCHULTE
KTH - Royal Institute of Technology, Stockholm
and
PETER J. STUCKEY
University of Melbourne

This paper explores the question of when two propagation-based constraint systems have the same behaviour, in terms of search space. We categorise the behaviour of domain and bounds propagators for primitive constraints, and provide theorems that allow us to determine propagation behaviours for conjunctions of constraints. We then show how we can use this to analyse CLP(FD) programs to determine when we can safely replace domain propagators by more efficient bounds propagators without increasing search space. Empirical evaluation shows that programs optimized by the analysis' results are considerably more efficient.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Constructs and Features—*Constraints*; D.3.3 [**Programming Languages**]: Language Classifications—*Constraint and logic languages*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Program analysis*

General Terms: Languages, Theory, Experimentation, Performance

Additional Key Words and Phrases: Constraint (logic) programming, finite domain constraints, bounds propagation, domain propagation, abstract interpretation, program analysis

1. INTRODUCTION

In building a finite domain constraint programming solution to a combinatorial problem a tradeoff arises in the choice of propagation that is used for each constraint: stronger propagation methods are more expensive to execute but may detect failure earlier; weaker propagation methods are (generally) cheaper to execute but may (exponentially) increase the search space explored to find an answer. In this paper we investigate the possibility of analysing finite domain constraint systems, or constraint programs, and determining whether the propagation methods used for some constraints could be replaced by simpler, and more efficient alternatives without increasing the size of the search space.

Consider the following example constraint where $x_1, \ldots, x_4$ range over integer

Author's addresses: C. Schulte, IMIT, KTH - Royal Institute of Technology, Isafjordsgatan 39, Electrum 229 SE-164 40 Kista, SWEDEN, email: schulte@imit.kth.se. P.J. Stuckey. Dept of Comp. Sci. & Soft. Eng., University of Melbourne 3010, AUSTRALIA, email: pjs@cs.mu.oz.au.

values 0 to 10:

$$x_1 \leq x_2, 2x_2 = 3x_3 + 1, x_3 \leq x_4$$

Each of the constraints could be implemented using domain propagation or bounds propagation. Clearly if each constraint is implemented using domain propagation we have stronger information, and the search space explored in order to find all solutions for the problem will be no larger than if we used bounds propagation. The question we ask is: can we get the same search space with bounds propagation?

Domain propagation on the constraints $x_1 \leq x_2$ and $x_3 \leq x_4$ is equivalent to bounds propagation since the constraints only place upper and lower bounds on their variables. This is not the case for $2x_2 = 3x_3 + 1$ where domain propagation reduces the domains (sets of possible values) of x_2 to $\{2, 5, 8\}$ and x_3 to $\{1, 3, 5\}$, while bounds propagation reduces x_2 to $\{2, 3, 4, 5, 6, 7, 8\}$ and x_3 to $\{1, 2, 3, 4, 5\}$. The question is: will execution require more search, if we use bounds propagation for this constraint as well?

Suppose that we use a labelling strategy that either assigns a variable to its lower bound, or constrains it to be greater than its lower bound. Then none of the constraints added during search creates holes in the domains. (This is in contrast to a strategy where we assign a variable to equal its middle value in its domain, or to exclude its middle value.) Hence the only holes in the domains of x_2 and x_3 will come from the constraint $2x_2 = 3x_3 + 1$. We will show that if the domains of x_2 and x_3 have no holes from other sources, domain propagation for $2x_2 = 3x_3 + 1$ fails iff bounds propagation fails. Hence the search space is the same for both bounds and domain propagation.

While for this simple example the advantage of bounds propagation over domain propagation may not seem significant, for more complex constraints there can be significant differences in efficiency of domain and bounds propagation. For example, domain propagation for `alldifferent` for n variables is $O(n^{2.5})$ [Régin 1994], while bounds propagation is $O(n \log n)$ [Puget 1998] and even $O(n)$ in common cases [Mehlhorn and Thiel 2000]. Similarly, domain propagation for a linear equation involving n variables is exponential while bounds propagation is $O(n)$.

In this paper we investigate when bounds and domain propagation will lead to the same search space.

The contributions of this paper are:

—We classify the behaviour of propagators for common primitive constraints, in particular introducing the crucial notion of *endpoint-relevant* propagators.

—We give theorems that allow us to extend reasoning about propagators for a single constraint to reasoning about propagators for a conjunction of constraints.

—We define an analysis algorithm for CLP(FD) programs that determines where we can replace domain propagators with bounds propagators without increasing the search space.

—We show examples where our analysis detects search space equivalent replacements and show the possible performance benefits that arise.

Previous authors [Mehlhorn and Thiel 2000; Puget 1998] have noted the difference in efficiency in bounds and domain propagation, for particular primitive constraints but not considered when different propagators lead to the same search space. The closest related work is [Harvey and Stuckey 2003], which considers the relative propagation strengths of different equivalent forms of constraints. Although both domain and bounds propagation are considered, bounds propagation is never compared to domain propagation.

Another somewhat related approach is to use type inference to derive properties of constraints [Lesaint 2002]. The framework starts from properties of primitive constraints and infers properties of conjunctions, disjunctions and negations of such constraints. In order to use the framework stability of properties under various operations needs to be established. It does not seem likely that the properties we introduce in the current paper fit the framework since stability under conjunction (see Theorem 3.29) has side conditions that do not seem to be expressible in the framework. Finally, [Lesaint 2002] does not mention how to actually take advantage of information derived by type inference.

While there has been considerable success in optimizing constraint programs over real linear constraints [Kelly et al. 1998], there has been little progress in optimizing finite domain CLP programs. Much of this stems from the difficulty in effectively analysing the behaviour of CLP(FD) solvers. In this paper we make a first step in this direction.

The remainder of the paper is organized as follows: in the next section we introduce terminology and define domain and bounds propagators. We then investigate properties of propagators and sets of propagators that allow us to prove search space equivalence. In Section 4, we define an analysis of CLP(FD) programs to gather information about propagation. We use this to define a program transformation that annotates individual constraints with the form of propagation we should use for them. Finally in Section 6 we conclude and give some directions for extending the work.

2. PROPAGATION BASED SOLVING

2.1 Basic Definitions

This paper considers integer constraint solving where Boolean variables are just considered as integer variables which range over values 0 (false) and 1 (true). We consider the following kinds of constraints.

—A *primitive linear constraint* is an equality ($=$), inequality ($\leq$), or disequation ($\neq$), written as $\sum_{i=1}^{n} a_i x_i \ \text{op}\ d$ where x_i are integer variables, a_i, d are integers, and $\text{op} \in \{=, \leq, \neq\}$.

—A *primitive reified constraint* is of the form $x_0 \Leftrightarrow \sum_{i=1}^{n} a_i x_i \ \text{op}\ d$ where x_i are integer variables, a_i, d are integers, and $\text{op} \in \{=, \leq, \neq\}$. These constraints are interpreted as x_0 is 1 if the linear constraint holds and 0 if the linear constraint does not hold.

—A *primitive Boolean constraint* is of the form $x_1 = \neg x_2$ (negation), $x_1 = (x_2 \,\&\&\, x_3)$ (conjunction), $x_1 = (x_2 \,||\, x_3)$ (disjunction), $x_1 = (x_2 \Rightarrow x_3)$ (implication), or $x_1 = (x_2 \Leftrightarrow x_3)$ (equivalence).

—A *primitive nonlinear constraint* is a multiplication $x_1 = x_2 \times x_3$, a squaring $x_1 = x_2 \times x_2$, a positive squaring $x_1 = x_2 \times x_2 \wedge x_2 \geq 0$, an absolute value constraint $x_1 = |x_2|$, a minimum constraint $x_1 = \mathtt{min}(x_2, x_3)$, an alldifferent constraint $\mathtt{alldifferent}([x_1, \ldots, x_n])$, or a default constraint $\mathtt{default}([x_1, \ldots, x_n])$.

A default constraint represents a nonlinear constraint with no further information on its constraint propagation available.

A *constraint* is a conjunction of primitive constraints, which we will sometimes treat as a set of primitive constraints.

Note that these primitive constraints include almost all integer constraints used in constraint programming systems. More complex constraints such as $x_1 + |x_2| \times (x_3)^2 \neq 3$ are broken down into conjunctions of these primitive constraints in constraint programming systems. For this example, it would be treated as $t_1 = |x_2| \wedge t_2 = x_3 \times x_3 \wedge t_3 = t_1 \times t_2 \wedge x_1 + t_3 \neq 3$ where t_1, t_2 and t_2 are new variables.

We use the notation $[x_1, \ldots, x_n] :: [l \mathinner{..} u]$ as shorthand for the conjunction of inequalities

$$x_1 \geq l, x_1 \leq u, \ldots, x_n \geq l, x_n \leq u$$

As additional shorthands, we use $\sum_{i=1}^{n} a_i x_i \geq d$ for $\sum_{i=1}^{n} -a_i x_i \leq -d$ as well as $x_0 \Leftrightarrow \sum_{i=1}^{n} a_i x_i \geq d$ for $x_0 \Leftrightarrow \sum_{i=1}^{n} -a_i x_i \leq -d$.

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 a constraint c, if $\mathcal{Z} \models_\theta c$ (resp. $\mathcal{R} \models_\theta c$).

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 with $D(x) = \emptyset$ for some x. The *intersection* of two domains D_1 and D_2, denoted $D_1 \sqcap D_2$, is defined as the domain $D(x) = D_1(x) \cap D_2(x)$ 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 $D_1(x) \subseteq D_2(x)$ (resp. $D_1(x) = D_2(x)$) for all $x \in V$.

In an abuse of notation, we define a valuation θ to be an element of a domain D, written $\theta \in D$, if $\theta(x) \in D(x)$ for all $x \in vars(\theta)$. 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 *propagator* f for a variable x is a function mapping a domain D to a set of values representing the possible values for x. A propagator only considers part of the domain corresponding to some subset of variables of interest which we denote by $vars(f)$.

We can extend propagators f for a variable x to map a domain D to another domain D'. Let $prop(f, D)$ denote the extension of f to map domains to domains, defined by $D'(x') = D(x')$ for $x' \neq x$, and $D'(x) = D(x) \cap f(D)$. Note that this extension guarantees $prop(f, D) \sqsubseteq D$ for any domain D.

A propagator f is correct for a constraint c, iff

$$\{\theta \in D \mid \mathcal{Z} \models_\theta c\} = \{\theta \in prop(f, D) \mid \mathcal{Z} \models_\theta c\}$$

EXAMPLE 2.1. For the constraint $c \equiv x_1 \geq x_2 + 1$ the function

$$f(D) = \{d \in D(x_1) \mid d \geq \inf_D x_2 + 1\}$$

is a correct propagator for variable x_1.

Let $D_1(x_1) = D_1(x_2) = \{1, 5, 8\}$, then $f(D_1) = \{5, 8\}$ and $prop(f, D_1) = D_2$ where $D_2(x_1) = \{5, 8\}$ and $D_2(x_2) = \{1, 5, 8\}$. □

A *propagation solver* $solv(F, D)$ for a set of propagators F and an initial domain D repeatedly applies all the propagators in F starting from domain D until there is no further change in the resulting domain. In other words, $solv(F, D)$ returns a new domain defined by

$$\begin{aligned} iter(F, D) &= \bigsqcap_{f \in F} prop(f, D) \\ solv(F, D) &= \mathrm{gfp}(\lambda d.\, iter(F, d))(D). \end{aligned}$$

where gfp denotes the greatest fixpoint w.r.t. $\sqsubseteq$ lifted to functions.

2.2 Domain-Consistent Propagators

A domain D is *domain-consistent* for a constraint c, if D is the least domain containing all integer solutions $\theta \in D$ of c, i.e, there does not exist $D' \sqsubset D$ such that $\theta \in D \wedge \mathcal{Z} \models_\theta c \Rightarrow \theta \in D'$.

A propagator set F maintains *domain-consistency* for a constraint c, if $solv(F, D)$ is always domain-consistent for c.

Define the *domain-consistency propagator* for a constraint c and a variable x, $dom(c, x)$, as follows

$$dom(c, x)(D) = \{\theta(x) \mid \theta \in D \text{ and } \theta \text{ is a solution of } c, \mathcal{Z} \models_\theta c\}.$$

Example 2.2. Consider the constraint $c \equiv x_1 = 3x_2 + 5x_3$ and the domain $D(x_1) = \{2, 3, 4, 5, 6, 7\}$, $D(x_2) = \{0, 1, 2\}$, and $D(x_3) = \{-1, 0, 1, 2\}$.

The solutions of c are

$$\{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 0\}, \{x_1 \mapsto 5, x_2 \mapsto 0, x_3 \mapsto 1\}, \{x_1 \mapsto 6, x_2 \mapsto 2, x_3 \mapsto 0\}$$

Hence, $iter(\{dom(c, x_1), dom(c, x_2), dom(c, x_3)\}, D)$ gives a domain D' such that $D'(x_1) = \{3, 5, 6\}$, $D'(x_2) = \{0, 1, 2\}$, and $D'(x_3) = \{0, 1\}$. D' is domain-consistent with respect to c, hence also $solv(\{dom(c, x_1), dom(c, x_2), dom(c, x_3)\}, D) = D'$. □

2.3 Ranges and Bounds Consistent Propagators

A *range* of integers $[l \,..\, u]$ is the set of integers $\{d \in \mathcal{Z} \mid l \leq d \leq u\}$. A domain is a *range domain* if $D(x)$ is a range for all x. Let $D' = range(D)$ be the smallest range domain containing D, i.e. domain $D'(x) = [\inf_D x \,..\, \sup_D x]$ for all x. A domain D_1 is *bounds-stronger* than a domain D_2, written $D_1 \overset{b}{\sqsubseteq} D_2$, if $range(D_1) \sqsubseteq range(D_2)$. Two domains D_1 and D_2 are *bounds-equal*, denoted $D_1 \overset{b}{\equiv} D_2$, if $range(D_1) = range(D_2)$.

There are two different definitions of bounds consistency used in the literature. We will call them bounds($\mathcal{R}$)-consistency, used by e.g. [Marriott and Stuckey 1998; Harvey and Schimpf 2002; Zhang and Yap 2000], and bounds($\mathcal{Z}$)-consistency, used by e.g. [Van Hentenryck et al. 1998; Puget 1998; Régin and Rueher 2000; Quimper et al. 2003]. Apt [Apt 2003] gives both definitions calling the first one bounds consistency, and the second interval consistency.

A domain D is *bounds*($\mathcal{R}$)*-consistent* for a constraint c and a variable x_i with $vars(c) = \{x_1, \ldots, x_n\}$, if for each $d_i \in \{\inf_D x_i, \sup_D x_i\}$ there exist *real numbers* d_j with $\inf_D x_j \leq d_j \leq \sup_D x_j$, $1 \leq j \neq i \leq n$ such that $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$ is a *real* solution of c. A domain D is *bounds*($\mathcal{R}$)*-consistent* for a constraint c, if it is bounds($\mathcal{R}$)-consistent for c and each $x \in vars(c)$.

A domain D is *bounds*($\mathcal{Z}$)*-consistent* for a constraint c and a variable x_i with $vars(c) = \{x_1, \ldots, x_n\}$, if for each $d_i \in \{\inf_D x_i, \sup_D x_i\}$ there exist *integer numbers* d_j with $\inf_D x_j \leq d_j \leq \sup_D x_j$, $1 \leq j \neq i \leq n$ such that $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$ is a *integer* solution of c. A domain D is *bounds*($\mathcal{Z}$)*-consistent* for a constraint c, if it is bounds($\mathcal{Z}$)-consistent for c and each $x \in vars(c)$.

A propagator set F maintains *bounds*(α)*-consistency* for a constraint c, if $solv(F, D)$ is always bounds(α)-consistent for c.

In this paper we will concentrate on bounds($\mathcal{R}$)-consistency, since it is weaker than bounds($\mathcal{Z}$)-consistency, and corresponds to the consistency implemented for most primitive constraints. If we can show domain-consistent propagators and bounds($\mathcal{R}$)-consistent propagators lead to equivalent search space, then this automatically extends to the stronger bounds($\mathcal{Z}$)-consistent propagators.

The *bounds*($\mathcal{R}$) *propagator* $bnd(c, x)$ for a primitive constraint c and variable x are defined as below.

Primitive linear constraints

—if $c \equiv \sum_{i=1}^{n} a_i x_i = d$, then for $1 \leq j \leq n$

$$bnd(c, x_j)(D) = [\lceil l \rceil \ .. \ \lfloor u \rfloor]$$

where

$$l = \inf_D \left(\frac{d - \sum_{i=1, i \neq j}^{n} a_i x_i}{a_j} \right) \quad \text{and} \quad u = \sup_D \left(\frac{d - \sum_{i=1, i \neq j}^{n} a_i x_i}{a_j} \right)$$

—if $c \equiv \sum_{i=1}^{n} a_i x_i \leq d$, then for $1 \leq j \leq n$

—if $a_j > 0$, then

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

—if $a_j < 0$, then

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

—if $c \equiv \sum_{i=1}^{n} a_i x_i \neq d$, then for $1 \leq j \leq n$

$$bnd(c, x_j)(D) = \begin{cases} [l + 1 \ .. \ u] & \text{if } a_j l = d - \sum_{i=1, i \neq j}^{n} a_i d_i \\ & \quad \text{where } D(x_i) = \{d_i\}, 1 \leq i \leq n, i \neq j \\ [l \ .. \ u - 1] & \text{if } a_j u = d - \sum_{i=1, i \neq j}^{n} a_i d_i \\ & \quad \text{where } D(x_i) = \{d_i\}, 1 \leq i \leq n, i \neq j \\ [l \ .. \ u] & \text{otherwise} \end{cases}$$

where $l = \inf_D x_j$ and $u = \sup_D x_j$.

Primitive reified constraints

—if $c \equiv x_0 \Leftrightarrow \sum_{i=1}^{n} a_i x_i = d$, then

—if $1 \leq j \leq n$, then

$$bnd(c, x_j)(D) = \begin{cases} bnd(\sum_{i=1}^{n} a_i x_i = d, x_j)(D) & \text{if } D(x_0) = \{1\} \\ bnd(\sum_{i=1}^{n} a_i x_i \neq d, x_j)(D) & \text{if } D(x_0) = \{0\} \\ [\inf_D x_j \mathrel{..} \sup_D x_j] & \text{otherwise} \end{cases}$$

—otherwise

$$bnd(c, x_0)(D) = \begin{cases} \{0\} & \text{if } \sum_{i=1}^{n} \inf_D(a_i x_i) > d \\ \{0\} & \text{if } \sum_{i=1}^{n} \sup_D(a_i x_i) < d \\ \{1\} & \text{if } \sum_{i=1}^{n} a_i d_i = d \\ & \quad \text{where } D(x_i) = \{d_i\}, 1 \leq i \leq n \\ [0 \mathrel{..} 1] & \text{otherwise} \end{cases}$$

—if $c \equiv x_0 \Leftrightarrow \sum_{i=1}^{n} a_i x_i \leq d$, then

—if $1 \leq j \leq n$, then

$$bnd(c, x_j)(D) = \begin{cases} bnd(\sum_{i=1}^{n} a_i x_i \leq d, x_j)(D) & \text{if } D(x_0) = \{1\} \\ bnd(\sum_{i=1}^{n} a_i x_i \geq d + 1, x_j)(D) & \text{if } D(x_0) = \{0\} \\ [\inf_D x_j \mathrel{..} \sup_D x_j] & \text{otherwise} \end{cases}$$

—otherwise

$$bnd(c, x_0)(D) = \begin{cases} \{0\} & \text{if } \sum_{i=1}^{n} \inf_D(a_i x_i) > d \\ \{1\} & \text{if } \sum_{i=1}^{n} \sup_D(a_i x_i) \leq d \\ [0 \mathrel{..} 1] & \text{otherwise} \end{cases}$$

—if $c \equiv x_0 \Leftrightarrow \sum_{i=1}^{n} a_i x_i \neq d$, then

—if $1 \leq j \leq n$, then

$$bnd(c, x_j)(D) = \begin{cases} bnd(\sum_{i=1}^{n} a_i x_i \neq d, x_j)(D) & \text{if } D(x_0) = \{1\} \\ bnd(\sum_{i=1}^{n} a_i x_i = d, x_j)(D) & \text{if } D(x_0) = \{0\} \\ [\inf_D x_j \mathrel{..} \sup_D x_j] & \text{otherwise} \end{cases}$$

—otherwise

$$bnd(c, x_0)(D) = \begin{cases} \{1\} & \text{if } \sum_{i=1}^{n} \inf_D(a_i x_i) > d \\ \{1\} & \text{if } \sum_{i=1}^{n} \sup_D(a_i x_i) < d \\ \{0\} & \text{if } \sum_{i=1}^{n} a_i d_i = d \\ & \quad \text{where } D(x_i) = \{d_i\}, 1 \leq i \leq n \\ [0 \mathrel{..} 1] & \text{otherwise} \end{cases}$$

Primitive Boolean constraints

—if $c \equiv x_1 = \neg x_2$, then for $1 \leq i \leq 2$

$$bnd(c, x_i)(D) = bnd(x_1 + x_2 = 1, x_i)(D) \cap [0 \mathrel{..} 1]$$

—if $c \equiv x_1 = (x_2 \mathbin{\&\&} x_3)$, then for $1 \leq i \leq 3$

$$bnd(c, x_i)(D) = bnd(x_1 \Leftrightarrow x_2 + x_3 = 2, x_i)(D) \cap [0 \mathrel{..} 1]$$

—if $c \equiv x_1 = (x_2 \mathbin{||} x_3)$, then for $1 \leq i \leq 3$

$$bnd(c, x_i)(D) = bnd(x_1 \Leftrightarrow x_2 + x_3 \geq 1, x_i)(D) \cap [0 \mathrel{..} 1]$$

—if $c \equiv x_1 = (x_2 \Rightarrow x_3)$, then for $1 \leq i \leq 3$

$$bnd(c, x_i)(D) = bnd(x_1 \Leftrightarrow x_2 - x_3 \leq 0, x_i)(D) \cap [0 \mathrel{..} 1]$$

—if $c \equiv x_1 = (x_2 \Leftrightarrow x_3)$, then for $1 \leq i \leq 3$

$$bnd(c, x_i)(D) = bnd(x_1 \Leftrightarrow x_2 - x_3 = 0, x_i)(D) \cap [0 \,..\, 1]$$

Primitive nonlinear constraints

—if $c \equiv x_1 = x_2 \times x_3$, then

— $bnd(c, x_1)(D) = [\inf E_1 \,..\, \sup E_1]$

where

$$E_1 = \{\ \inf_D x_2 \times \inf_D x_3,\ \ \inf_D x_2 \times \sup_D x_3,\ \sup_D x_2 \times \inf_D x_3,\ \sup_D x_2 \times \sup_D x_3\}$$

— $$bnd(c, x_2)(D) = \begin{cases} [\inf_D x_2 \,..\, \sup_D x_2] & \text{if } 0 \in [\inf_D x_3 \,..\, \sup_D x_3] \\ & \quad \text{and } 0 \in [\inf_D x_1 \,..\, \sup_D x_1] \\ [\lceil \inf E_2 \rceil \,..\, \lfloor \sup E_2 \rfloor] & \text{if } \inf_D x_3 > 0 \text{ or } \sup_D x_3 < 0 \\ [\inf R \,..\, \sup R] & \text{otherwise} \end{cases}$$

where

$$E_2 = \{\ \inf_D x_1 / \inf_D x_3,\ \ \inf_D x_1 / \sup_D x_3,\ \sup_D x_1 / \inf_D x_3,\ \sup_D x_1 / \sup_D x_3\}$$

and

$$R = \bigcup_{m \in \{1,2\}} (bnd(c, x_2)(D_m) \cap [\inf_D x_2 \,..\, \sup_D x_2])$$

with

$$\begin{array}{lll} D_1(x_1) = D(x_1), & D_1(x_2) = D(x_2), & D_1(x_3) = [1 \,..\, \sup_D x_3] \\ D_2(x_1) = D(x_1), & D_2(x_2) = D(x_2), & D_2(x_3) = [\inf_D x_3 \,..\, -1] \end{array}$$

—The propagator for $bnd(c, x_3)$ is defined analogously to $bnd(c, x_2)$.

—if $c \equiv x_1 = x_2 \times x_2$, then

— $$bnd(c, x_1)(D) = \begin{cases} \left[(\inf_D x_2)^2 \,..\, (\sup_D x_2)^2\right] & \text{if } \inf_D x_2 \geq 0 \\ \left[(\sup_D x_2)^2 \,..\, (\inf_D x_2)^2\right] & \text{if } \sup_D x_2 \leq 0 \\ \left[0 \,..\, \sup\{(\inf_D x_2)^2, (\sup_D x_2)^2\}\right] & \text{otherwise} \end{cases}$$

— $$bnd(c, x_2)(D) = \begin{cases} \left[\left\lceil \sqrt{\inf_D x_1} \right\rceil \,..\, \left\lfloor \sqrt{\sup_D x_1} \right\rfloor\right] & \text{if } \inf_D x_2 \geq 0 \\ \left[\left\lceil -\sqrt{\sup_D x_1} \right\rceil \,..\, \left\lfloor -\sqrt{\inf_D x_1} \right\rfloor\right] & \text{if } \sup_D x_2 \leq 0 \\ [\inf R \,..\, \sup R] & \text{otherwise} \end{cases}$$

where

$$R = (bnd(c, x_2)(D_1) \cup bnd(c, x_2)(D_2)) \cap [\inf_D x_2 \,..\, \sup_D x_2]$$

with

$$\begin{array}{ll} D_1(x_1) = D(x_1), & D_1(x_2) = [0 \,..\, \sup_D x_2] \\ D_2(x_1) = D(x_1), & D_2(x_2) = [\inf_D x_2 \,..\, 0] \end{array}$$

—if $c \equiv x_1 = x_2 \times x_2 \wedge x_2 \geq 0$, then for $1 \leq i \leq 2$

$$bnd(c, x_i)(D) = bnd(x_1 = x_2 \times x_2, x_i)(D) \cap [0 \,..\, \sup_D x_i]$$

—if $c \equiv x_1 = |x_2|$, then

— $bnd(c, x_1)(D) = \begin{cases} [\inf_D x_2 \,..\, \sup_D x_2] & \text{if } \inf_D x_2 \geq 0 \\ [\sup_D x_2 \,..\, \inf_D x_2] & \text{if } \sup_D x_2 \leq 0 \\ [0 \,..\, \sup\{\inf_D x_2, \sup_D x_2\}] & \text{otherwise} \end{cases}$

— $bnd(c, x_2)(D) = \begin{cases} [\inf_D x_1 \,..\, \sup_D x_1] & \text{if } \inf_D x_2 \geq 0 \\ [\sup_D x_1 \,..\, \inf_D x_1] & \text{if } \sup_D x_2 \leq 0 \\ [\inf R \,..\, \sup R] & \text{otherwise} \end{cases}$

where

$$R = (bnd(c, x_2)(D_1) \cup bnd(c, x_2)(D_1)) \cap [\inf_D x_2 \,..\, \sup_D x_2]$$

with

$$\begin{array}{rclrcl} D_1(x_1) & = & D(x_1), & D_1(x_2) & = & [0 \,..\, \sup_D x_2] \\ D_2(x_1) & = & D(x_1), & D_2(x_2) & = & [\inf_D x_2 \,..\, 0] \end{array}$$

—if $c \equiv x_1 = \mathtt{min}(x_2, x_3)$, then

— $bnd(c, x_1)(D) = [\inf\{\inf_D x_2, \inf_D x_3\} \,..\, \inf\{\sup_D x_2, \sup_D x_3\}]$

— $bnd(c, x_2)(D) = \begin{cases} [\inf_D x_1 \,..\, \sup_D x_1] & \text{if } \sup_D x_2 \leq \inf_D x_3 \text{ or } \sup_D x_1 < \inf_D x_3 \\ [\inf_D x_1 \,..\, \sup_D x_2] & \text{otherwise} \end{cases}$

—The propagator for $bnd(c, x_3)$ is defined analogously to $bnd(c, x_2)$.

Example 2.3. Consider the same constraint $c \equiv x_1 = 3x_2 + 5x_3$ and domain $D(x_1) = \{2, 3, 4, 5, 6, 7\}$, $D(x_2) = \{0, 1, 2\}$, and $D(x_3) = \{-1, 0, 1, 2\}$ as in Example 2.2.

Calculation of

$$D' = iter(\{bnd(c, x_1), bnd(c, x_2), bnd(c, x_3)\}, D)$$

determines that

$$\begin{array}{rcl} D'(x_1) & = & [l_1 \,..\, u_1] = [2 \,..\, 7] \\ & & \text{with } \; l_1 \; = \sup\left\{\left\lceil \dfrac{0 + 3 \times 0 + 5 \times -1}{1} \right\rceil, 2\right\} \\ & & \text{and } \; u_1 \; = \inf\left\{\left\lfloor \dfrac{0 + 3 \times 2 + 5 \times 2}{1} \right\rfloor, 7\right\} \end{array}$$

and

$$\begin{array}{rcl} D'(x_3) & = & [l_3 \,..\, u_3] = [0 \,..\, 1] \\ & & \text{with } \; l_3 \; = \sup\left\{\left\lceil \dfrac{0 - 3 \times 2 + 1 \times 2}{5} \right\rceil, -1\right\} \\ & & \text{and } \; u_3 \; = \inf\left\{\left\lfloor \dfrac{0 - 3 \times 0 + 1 \times 7}{5} \right\rfloor, 2\right\} \end{array}$$

While the domain of x_3 is modified, the domains of x_1 and x_2 remain unchanged. The resulting domain D' is bounds($\mathcal{R}$)-consistent with the constraint c.

Notice that bounds propagation has determined less information than domain propagation. In particular also less information about bounds has been determined: bounds propagation computes $\inf_{D'} x_1$ to be 2 as opposed to 3 obtained by domain-propagation in Example 2.2. □

THEOREM 2.4. *For all c, the set of propagators*

$$\{bnd(c,x) \mid x \in vars(c)\}$$

defined above maintains bounds($\mathcal{R}$)-consistency for c.

PROOF. For each c we will assume that $F = \{bnd(c,x) \mid x \in vars(c)\}$ and $D = solv(F, D_0)$. We use the notation $l_i = \inf_D x_i$ and $u_i = \sup_D x_i$. We show that each bound l_i, and u_i is bounds consistent for c.

Primitive linear constraints

—For $c \equiv \sum_{i=1}^{n} a_i x_i \leq d$ a proof is given in [Zhang and Yap 2000].

—For $c \equiv \sum_{i=1}^{n} a_i x_i = d$. We assume for simplicity $a_i > 0$ for $1 \leq i \leq n$, the other cases are similar. Consider w.l.o.g. l_1 and u_1. Now by the definition of $bnd(c, x_1)$

$$\frac{d - \sum_{i=2}^{n} a_i \sup_D x_i}{a_1} \leq l_1 \leq u_1 \leq \frac{d - \sum_{i=2}^{n} a_i \inf_D x_i}{a_1}$$

Hence there is clearly a solution $\theta = \{x_1 \mapsto l_1\} \cup \{x_i \mapsto d_i \mid 2 \leq i \leq n\}$ of c where $l_i \leq d_i \leq u_i$ for $2 \leq i \leq n$. Similarly there is a solution for $x_1 \mapsto u_1$.

—For $c \equiv \sum_{i=1}^{n} a_i x_i \neq d$. Consider w.l.o.g. l_1 and u_1. Now if $|D(x_i)| \geq 2$ for some $2 \leq i \leq n$ then there are clearly solutions to c with $x_1 \mapsto l_1$ and $x_1 \mapsto u_1$. Otherwise, $|D(x_i)| = \{d_i\}$ for all $2 \leq i \leq n$. Then by the definition of $bnd(c, x_1)$ we have that $a_1 l_1 \neq d - \sum_{i=2}^{n} a_i d_i$ and $a_1 u_1 \neq d - \sum_{i=2}^{n} a_i d_i$. Hence l_1 and u_1 are bounds consistent.

Primitive reified constraints. For a reified linear constraint $c \equiv (x_o \Leftrightarrow c')$. If $l_0 = 1$ or $u_0 = 0$ then the bounds consistency follows from the correctness of the appropriate linear constraint c'. If $l_0 = 0$ and $u_0 = 1$ then clearly the bounds for the variables x_i for $1 \leq i \leq n$ are bounds consistent, since we can either satisfy or disatisfy the constraint c'.

We now address the bounds consistency of l_0 and u_0 themselves. For $l_0 = 0$ to be bounds consistent we need to find a valuation which disatisfies c'. Now the only cases where this is impossible is where each θ such that $l_i \leq \theta(x_i) \leq u_i$ for $1 \leq i \leq n$ satisfies c'. If there is no such valuation, this is captured by $bnd(c, x_0)$ and yields $D(x_0) = \{1\}$. This contradicts $l_0 = 1$. Similarly for $u_0 = 1$ to be bounds consistent we need to find a valuation θ such that $l_i \leq \theta(x_i) \leq u_i$ for $1 \leq i \leq n$ which satisfies c'. Again if there is no such valuation this is captured by $bnd(c, x_0)$ and yields $D(x_0) = \{0\}$. This contradicts $u_0 = 0$.

Primitive Boolean constraints. For Boolean constraints the proof follows directly from the correctness of reified linear constraints, and the correctness of the modelling of the Boolean constraints as reified linear constraints.

Primitive nonlinear constraints

—For $c \equiv x_1 = x_2 \times x_3$. We first show that l_1 and u_1 are bounds consistent. Suppose $a_2 \times a_3 = \inf E_1$ and $b_2 \times b_3 = \sup E_1$ (see the definition of $bnd(c, x_1)$ for E_1), where $\{a_i, b_i\} \subseteq \{l_i, u_i\}$ for $2 \leq i \leq 3$. Now $a_2 \times a_3 \leq l_1 \leq u_1 \leq b_2 \times b_3$. Due to the continuity of multiplication, there exists $d_i \in [a_i \mathinner{..} b_i] \cup [b_i \mathinner{..} a_i]$ for $2 \leq i \leq 3$ such that $l_1 = d_2 \times d_3$. And clearly $l_i \leq d_i \leq u_i$. Hence l_1 is bounds consistent. Similarly for u_1.

Now we consider l_2 and u_2, the case for l_3 and u_3 is analogous. If $l_3 \leq 0 \leq u_3$ and $l_1 \leq 0 \leq u_1$ then l_2 and u_2 are bounds consistent setting the other variables to 0. Suppose $l_3 > 0$. Then $l_2 \geq l_1/l_3$ and $u_2 \leq u_1/l_3$ by definition of E_2. Since $l_1/l_3 \leq l_2 \leq u_3 \leq u_1/l_3$ we have a solution $l_2 = d/l_3$ and $u_2 = d'/l_3$ where $l_1 \leq d \leq d' \leq u_1$ which shows that l_2 and u_2 are bounds consistent. We can similarly argue if $u_3 < 0$.

Otherwise $l_3 \leq 0 \leq u_3$ and either $l_1 > 0$ or $u_1 < 0$. We argue the case for $l_1 > 0$, the other case is similar. We break the domain of x_3 into two parts containing the negative and the positive values. Note that the value 0 for x_3 cannot lead to a solution consistent with the domain of x_1.

Clearly $\inf R \geq l_2$ by definition but also $l_2 \geq \inf R$ since this case of the definition of $bnd(c, x_2)$ applies. Hence $\inf R = l_2$ and similarly $\sup R = u_2$. Now since $l_1 > 0$, we have that $bnd(c, x_2)(D_1) = \left[\lceil \frac{l_1}{u_3} \rceil \mathinner{..} u_1\right]$ if $u_3 > 0$. Otherwise $bnd(c, x_2)(D_1)$ is empty. Similarly $bnd(c, x_2)(D_2) = \left[-u_1 \mathinner{..} \lfloor \frac{l_1}{l_3} \rfloor\right]$ if $l_3 < 0$ and otherwise it is empty. Since $l_2 = \inf R$ either $-u_1 \leq l_2 \leq \lfloor \frac{l_1}{l_3} \rfloor$ or $\lceil \frac{l_1}{u_3} \rceil \leq l_2 \leq u_1$. In the first case, by the continuity of multiplication, there exists $d_1 \in [l_1 \mathinner{..} u_1]$ and $d_3 \in [l_3 \mathinner{..} -1]$ such that $l_2 = d_1/d_3$. Hence l_2 is bounds consistent, similarly for the second case $d_1 \in [l_1 \mathinner{..} u_1]$ and $d_3 \in [1 \mathinner{..} u_3]$. Similar reasoning applies to show that u_2 is bounds consistent.

—For $c \equiv x_1 = x_2 \times x_2$. Suppose $l_2 \geq 0$ then $l_1 \geq (l_2)^2$ and $l_2 \geq \sqrt{l_1}$ and hence $l_1 = (l_2)^2$. Similarly for $u_1 = (u_2)^2$. Hence D is bounds consistent with c. Suppose $u_2 \leq 0$ then similarly $l_1 = (u_2)^2$ and $u_1 = (l_2)^2$ and D is bounds consistent with c.

Otherwise $l_2 < 0$ and $u_2 > 0$. Then we split the domain of x_2 into two parts D_1 and D_2 where $\inf_{D_1} x_2 = 0$ and $\sup_{D_2} x_2 = 0$. Now if $bnd(c, x_2)(D_1) \cap [l_2 \mathinner{..} u_2] = \emptyset$ then $l_2 \geq 0$ which is a contradiction. Similarly if $bnd(c, x_2)(D_2) \cap [l_2 \mathinner{..} u_2] = \emptyset$ then $u_2 \leq 0$ which again is a contradiction. Hence $-\sqrt{u_1} \leq l_2 \leq -\sqrt{l_1}$ and $\sqrt{l_1} \leq u_2 \leq \sqrt{u_1}$. Hence both l_2 and u_2 are bounds consistent. We also have that $u_1 \leq \sup\{(l_2)^2, (u_2)^2\}$ which together means either $u_1 = (l_2)^2$ or $u_1 = (u_2)^2$. So u_1 is clearly bounds consistent. Now l_1 is bounds consistent since either $l_2 = -\sqrt{u_1} \leq -\sqrt{l_1}$ or $\sqrt{l_1} \leq \sqrt{u_1} = u_2$.

—For $c \equiv x_1 = x_2 \times x_2 \wedge x_2 \geq 0$ the proof for $x_1 = x_2 \times x_2$ suffices.

—For $c \equiv x_1 = |x_2|$ the proof is almost identical to that for $x_1 = x_2 \times x_2$.

—For $c \equiv x_1 = \min(x_2, x_3)$. We have that $l_1 \geq \inf(l_2, l_3)$ and $l_2 \geq l_1$ and $l_3 \geq l_1$. Hence $l_1 = l_2$ or $l_1 = l_3$. The solution $\{x_1 \mapsto l_1, x_2 \mapsto l_2, x_3 \mapsto l_3\}$ proves that the lower bounds are bounds consistent. Now $u_1 \leq u_2$ and $u_1 \leq u_3$. If $u_1 < l_3$ then $u_2 \leq u_1$ and hence $u_1 = u_2$. In this case the solution $\{x_1 \mapsto u_1, x_2 \mapsto u_2, x_3 \mapsto u_3\}$ proves the upper bounds are bounds consistent. Otherwise $u_1 \geq l_3$ and the solution $\{x_1 \mapsto u_1, x_2 \mapsto u_2, x_3 \mapsto u_1\}$ show that the upper bounds of x_1 and x_2 are bounds consistent. Similarly if $u_1 < l_2$ then $u_3 = u_1$ and the solution $\{x_1 \mapsto u_1, x_2 \mapsto u_2, x_3 \mapsto u_3\}$ proves the upper bounds are bounds consistent. Otherwise $u_1 \geq l_2$ and the solution $\{x_1 \mapsto u_1, x_2 \mapsto u_1, x_3 \mapsto u_3\}$ show that the upper bounds of x_1 and x_3 are bounds consistent.

□

While for almost all propagators we consider bounds($\mathcal{R}$)-consistency the exception is the `alldifferent` constraint. Here the bounds propagators defined in Puget [1998] or Mehlhorn and Thiel [2000] maintain bounds($\mathcal{Z}$)-consistency. We let $bnd(c, x)$ where c is $\texttt{alldifferent}([x_1, \ldots, x_n])$ be the bounds($\mathcal{Z}$) propagator defined as in [Puget 1998] or [Mehlhorn and Thiel 2000].

3. CATEGORISING PROPAGATORS

In order to reason about the propagation behaviour of propagators corresponding to primitive constraints, we need to be able to categorise their behaviour. In order for bounds propagation to be as powerful as domain propagation we will need to understand how individual propagators relate to bounds.

Definition 3.1. A propagator f is *bounds-only*, if $f(D)$ is a range for all domains D.

A propagator f is *bounds-preserving*, if for all domains D such that $D(x)$ is a range for all $x \in vars(f)$, then $f(D)$ is a range.

Example 3.2. Clearly all bounds propagators are bounds-only and thus also bounds-preserving. Typically, domain propagators are not bounds-preserving, for example $dom(x_1 = 2x_2, x_1)$ is not bounds-preserving.

Some domain propagators are however bounds-preserving, for example $dom(x_1 = 2x_2, x_2)$, or $dom(x_1 = x_2 + 3, x_1)$ as well as $dom(x_1 = x_2 + 3, x_2)$. □

Example 3.3. Note that propagation is highly dependent on the nature of the constraints. For example, if $c_1 \equiv x_1 \geq 3x_2$ and $c_2 \equiv x_1 \leq 3x_2 + 1$, then $dom(c_1, x_1)$ and $dom(c_2, x_1)$ are both bounds-only. But the domain propagator $dom(c_1 \wedge c_2, x_1)$ on x_1 for the combined constraint $c_1 \wedge c_2$ is not bounds-only.

For example, if $D(x_1) = D(x_2) = [0 \,..\, 8]$ and $D' = prop(dom(c_1 \wedge c_2, x_1), D)$ we have that $D'(x_1) = \{0, 1, 3, 4, 6, 7\}$. □

3.1 Equivalence and Bounds-Equivalence

In order to replace one set of propagators by another we need to have notions of equivalence between sets of propagators.

Definition 3.4. Two sets of propagators F_1 and F_2 are *equivalent*, if for each domain D, $solv(F_1, D) = solv(F_2, D)$.

Equivalent sets of propagators of course can be used to replace each other in any context. Clearly a bounds propagator and a domain propagator will rarely be equivalent, since the domain propagator will remove values from inside domains. Hence we introduce bounds-equivalence.

Definition 3.5. Two sets of propagators F_1 and F_2 are *bounds-equivalent*, iff for each domain D, $solv(F_1, D) \stackrel{b}{\equiv} solv(F_2, D)$. That is, the resulting domains have the same endpoints for each variable.

The key to ensuring that two sets of propagators lead to the same search space is the following obvious result.

Proposition 3.6. *Let F_1 and F_2 be two bounds-equivalent sets of propagators. For any domain D, then $solv(F_1, D)$ is a false domain iff $solv(F_2, D)$ is a false domain.* □

With respect to search, this proposition can be interpreted as follows. Bounds-equivalent sets of propagators lead to the same failed nodes. We will consider two parallel executions where at each stage the set of propagators in each execution are bounds-equivalent. This means that one execution fails iff the other execution fails. We defer the full discussion of this to Section 4.

We are now in a position to examine the domain and bounds propagators for individual primitive constraints and determine relationships between them. The first lemma is obvious, its proof can be found in [Zhang and Yap 2000].

LEMMA 3.7. *Let* $c \equiv \Sigma_{i=1}^{n} a_i x_i \leq d$. *Then* $\{dom(c, x_i)\}$ *and* $\{bnd(c, x_i)\}$ *are equivalent for* $1 \leq i \leq n$. □

LEMMA 3.8. *Let* $c \equiv x_0 \Leftrightarrow \Sigma_{i=1}^{n} a_i x_i \leq d$. *Then* $\{dom(c, x_i)\}$ *and* $\{bnd(c, x_i)\}$ *are equivalent for* $0 \leq i \leq n$.

PROOF. First consider $x = x_i$ for some $1 \leq i \leq n$. If $D(x_0) = [0 .. 1]$ then $\{dom(c, x)\}$ and $\{bnd(c, x)\}$ both return $D(x)$ since the inequality could hold or not hold. If $D(x_0) = \{1\}$ then $dom(c, x)$ and $bnd(c, x)$ are equivalent respectively to $dom(c', x)$ and $bnd(c', x)$ where $c' \equiv \Sigma_{i=1}^{n} a_i x_i \leq d$ and hence by Lemma 3.7 they are equivalent. Similarly, when $D(x_0) = \{0\}$.

Now consider $x = x_0$. If there exists $\theta_1 \in D$ such that $\Sigma_{i=1}^{n} a_i \theta_1(x_i) \leq d$ and $\theta_2 \in D$ such that $\Sigma_{i=1}^{n} a_i \theta_2(x_i) > d$ then both $dom(c, x)$ and $bnd(c, x)$ return $[0 .. 1]$.

Otherwise assume $\Sigma_{i=1}^{n} a_i \theta(x_i) > d$ for all $\theta \in D$ and hence $dom(c, x)(D) = \{0\}$. Now by definition $\Sigma_{i=1}^{n} \inf_D(a_i x_i) = \inf\{\Sigma_{i=1}^{n} a_i \theta(x_i) \mid \theta \in D\}$. Hence $\Sigma_{i=1}^{n} \inf_D(a_i x_i) > d$ and thus $bnd(c, x)(D) = \{0\}$. Similar reasoning applies for the case where $\Sigma_{i=1}^{n} a_i \theta(x_i) \leq d$ for all $\theta \in D$. □

Bounds propagators and domain propagators for Boolean constraints are equivalent, essentially because a change to a variable domain also changes its bounds.

LEMMA 3.9. *Let* c *be a primitive Boolean constraint, and* $x \in vars(c)$. *Then* $\{dom(c, x)\}$ *and* $\{bnd(c, x)\}$ *are equivalent.*

PROOF. The proofs are simple case analyses, we illustrate the proof for $c \equiv x_1 = (x_2 \;\&\&\; x_3)$.

First consider x_1. Now $dom(c, x_1)(D)$ reduces the domain of x_1 in two cases, either (a) $D(x_2) = D(x_3) = \{1\}$ and it becomes $\{1\}$, or (b) $D(x_2) = \{0\}$ or $D(x_3) = \{0\}$ in which case it becomes $\{0\}$. Clearly for (a) the third case of the definition for $bnd(x_1 \Leftrightarrow x_2 + x_3 = 2, x_1)$ holds, and for (b) the second case holds.

Now consider x_2 (x_3 is symmetric). Now $dom(c, x_2)(D)$ reduces the domain of x_2 in two cases, either (c) $D(x_1) = \{1\}$ in which case it becomes $\{1\}$, or (d) $D(x_1) = \{0\}$ and $D(x_3) = \{1\}$ in which case it becomes $\{0\}$. Clearly for (c) $bnd(c, x_2)(D)$ is equivalent to $bnd(x_2 + x_3 = 2, x_2)(D)$. Using the first case for primitive linear constraints yields $l = 1$ (assuming $1 \in D(x_3)$, otherwise $l = 2$ and we get an empty domain). Thus the appropriate domain change occurs. For (d) $bnd(c, x_2)(D)$ is equivalent to $bnd(x_2 + x_3 \neq 2, x_2)(D)$ and since $D(x_3) = \{1\}$ the value 1 is removed from the domain of x_2. □

We should be careful, obviously not every constraint involving only Boolean variables is such that the domain and bounds propagators are equivalent. Consider

$c \equiv 2x_1 - 5x_2 + 7x_3 - 11x_4 + 13x_5 = 8$ where $D(x_i) = [0 \,..\, 1], 1 \leq i \leq 5$ then $prop(dom(c, x_1), D) = D'$ where $D'(x_1) = \{0\}$ while $prop(bnd(c, x_1), D) = D$.

Unfortunately bounds-equivalence by itself is not a strong enough notion since it does not hold that given F_1 and F_2 are bounds equivalents sets of propagators and F_1' and F_2' are bounds-equivalent that $F_1 \cup F_1'$ and $F_2 \cup F_2'$ are bounds-equivalent.

EXAMPLE 3.10. Consider the constraint $c(x_i, x_j)$ with solutions

$$\begin{array}{ll} \{x_i \mapsto 0, x_j \mapsto 1\} & \{x_i \mapsto 0, x_j \mapsto -1\} \\ \{x_i \mapsto 1, x_j \mapsto 0\} & \{x_i \mapsto -1, x_j \mapsto 0\} \end{array}$$

Also consider the propagator f_{ij} for variable x_j defined as

$$f_{ij}(D) = \begin{cases} [-1 \,..\, 1] & 0 \in D(x_i) \\ \{0\} & 0 \notin D(x_i), \{1, -1\} \cap D(x_i) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

Then $dom(c(x_i, x_j), x_j)$ and the propagator f_{ij} for variable x_j are bounds-equivalent. Hence $dom(c(x_1, x_2), x_2)$ and f_{12} are bounds-equivalent, and so are $dom(c(x_2, x_3), x_3)$ and f_{23}. But $\{dom(c(x_1, x_2), x_2), dom(c(x_2, x_3), x_3)\}$ and $\{f_{12}, f_{23}\}$ are not!

Consider $D(x_1) = \{0\}, D(x_2) = D(x_3) = [-1 \,..\, 1]$. Then $solv(\{f_{12}, f_{23}\}, D) = D$ while $solv(\{dom(c(x_1, x_2), x_2), dom(c(x_2, x_3), x_3)\}, D) = D'$ where $D'(x_1) = D'(x_3) = \{0\}$ and $D'(x_2) = \{-1, 1\}$. □

Hence we will need to introduce further classifications of propagators.

3.2 Endpoint-Relevance

In order to proceed we need to understand what propagators will give the same behaviour when applied to two bounds-equivalent domains. We introduce endpoint-relevance which captures the idea of a set of propagators in whose result the endpoints of the domain support each other, hence the parts of the domain except the endpoints are not relevant to the propagators bounds behaviour.

DEFINITION 3.11. A set of propagators F is *endpoint-relevant* if for all domains D, if $D_1 = solv(F, D)$ and $D_2 \overset{b}{\equiv} D_1$ then $solv(F, D_2) \overset{b}{\equiv} D_1$.

Note that, crucially, endpoint-relevant propagator sets only have special properties at fixpoints of the set of propagators. Otherwise the notion is too strong.

EXAMPLE 3.12. The set $\{dom(x_1 = 2x_2, x_2), dom(x_1 = 2x_2, x_1)\}$ is endpoint-relevant. Endpoint-relevance requires that endpoints are supported only by other endpoints. Note that $\{dom(x_1 = 2x_2, x_2)\}$ is not endpoint-relevant by itself, consider $D_1(x_1) = [1 \,..\, 7], D_1(x_2) = [1 \,..\, 3]$ and $D_2(x_1) = \{1, 3, 4, 5, 7\}, D_2(x_2) = [1 \,..\, 3]$. □

Because disequalities have very weak propagators there is a strong correspondence between their domain and bounds propagators.

LEMMA 3.13. *Let* $c \equiv \Sigma_{i=1}^n a_i x_i \neq d$. *Then* $dom(c, x_i)$ *and* $bnd(c, x_i)$ *are bounds-equivalent and endpoint-relevant for* $1 \leq i \leq n$.

PROOF. Both $dom(c, x_i)$ and $bnd(c, x_i)$ only depend on the endpoints of the input domain since they only remove a value d when each variable in $vars(c) - \{x_i\}$

has a fixed value (in which case the bounds are equal). Hence they are both endpoint-relevant.

The only difference between $dom(c, x_i)$ and $bnd(c, x_i)$ is when the non domain-consistent value d for x_i is neither a lower nor upper bound. In either case the resulting bounds do not change. □

Two variable equations are also endpoint-relevant because they only involve two variables, and there is a one-to-one correspondence between the values in any solution.

LEMMA 3.14. *Let c be a linear integer equation of the form $b_1x_1 + b_2x_2 = e$. Then $\{dom(c, x_1), dom(c, x_2)\}$ and $\{bnd(c, x_1), bnd(c, x_2)\}$ are bounds-equivalent and endpoint-relevant.*

PROOF. Assume w.l.o.g. that $b_1 > 0$. If this is not the case, we can replace x_1 by a new variable $-x_1'$ and assume $D(x_1') = \{-d \mid d \in D(x_1)\}$. Similarly, assume $b_2 > 0$.

Let $D_1 = solv(\{dom(c, x_1), dom(c, x_2)\}, D)$ the result of domain propagation and $D_2 = solv(\{bnd(c, x_1), bnd(c, x_2)\}, D)$ the result of bounds propagation. First by definition $D_1 = iter(\{dom(c, x_1), dom(c, x_2)\}, D)$. Hence

$$d_1 \in D_1(x_1) \quad \text{iff} \quad \frac{e - b_1d_1}{b_2} \in D(x_2) \tag{1}$$

$$d_2 \in D_1(x_2) \quad \text{iff} \quad \frac{e - b_2d_2}{b_1} \in D(x_1) \tag{2}$$

Clearly also $b_1 \inf_{D_1} x_1 + b_2 \sup_{D_1} x_2 = e$ and $b_1 \sup_{D_1} x_1 + b_2 \inf_{D_1} x_2 = e$. By the definition of bounds propagation we have that $b_1 \inf_{D_2} x_1 + b_2 \sup_{D_2} x_2 = e$ and $b_1 \sup_{D_2} x_1 + b_2 \inf_{D_2} x_2 = e$. This shows that both sets of propagators are endpoint-relevant.

Now because the endpoints match the conditions of (1) and (2) we have that $\{\inf_{D_2} x_1, \sup_{D_2} x_1\} \subseteq D_1(x_1)$ and similarly for x_2.

Let $D_2^0 = D$, $D_2^{2i+1} = iter(bnd(c, x_1), D_2^{2i})$, and $D_2^{2i+2} = iter(bnd(c, x_2), D_2^{2i+1})$ for $i \geq 0$. We show by induction that $\inf_{D_2^k} x_j \leq \inf_{D_1} x_j$ and $\sup_{D_2^k} x_j \geq \sup_{D_1} x_j$ for $j = 1, 2$ and $k \geq 0$. The base case is straightforward. Suppose $D_2^{k+1}(x_j) \neq D_2^k(x_j)$. We show that the result still holds for D_2^{k+1}. We consider the case when the lower bound of x_1 changes, the other cases are similar. The new lower bound is $\inf_{D_2^{k+1}} x_1 = \lceil \frac{e - b_2 \sup_{D_2^k} x_2}{b_1} \rceil$. Now by induction hypothesis $b_2 \sup_{D_2^k} x_2 \geq b_2 \sup_{D_1} x_2$ and $\inf_{D_1} x_1 = \frac{e - b_2 \sup_{D_1} x_2}{b_1}$ hence $\inf_{D_2^{k+1}} x_1 \leq \inf_{D_1} x_1$.

Finally there exists $k > 0$ such that $D_2^k = D_2$ by the definition of D_2. □

EXAMPLE 3.15. Perhaps surprisingly the domain and bounds propagators for reified constraints of the form $x_0 \Leftrightarrow a_1x_1 + a_2x_2 = d$ (or equivalently $x_0 \Leftrightarrow a_1x_1 + a_2x_2 \neq d$) are not bounds-equivalent. It would seem likely since if $x_0 = 1$ this acts like the constraint $a_1x_1 + a_2x_2 = d$ and if $x_0 = 0$ it acts like the constraint $a_1x_1 + a_2x_2 \neq d$ both of which have domain and bounds propagators that are bounds-equivalent.

The problem is when the constraints acts in the reverse direction (the domains of x_1 and x_2 affecting the domain of x_0). For example, given $c \equiv x_0 \Leftrightarrow x_1 -$

$x_2 = 0$ and domain $D(x_0) = [0\,..\,1]$, $D(x_1) = \{2,4,6\}$, and $D(x_2) = \{3,5,7\}$, the domain propagators F for c determines $\mathit{prop}(F,D)(x_0) = \{0\}$, while for the bounds propagators F' we have $\mathit{prop}(F',D)(x_0) = [0\,..\,1]$. □

Similarly to equations on two variables, positive squaring constraints are endpoint-relevant.

LEMMA 3.16. *Let c be $x_1 = x_2 \times x_2 \wedge x_2 \geq 0$. Then $\{\mathit{dom}(c,x_1), \mathit{dom}(c,x_2)\}$ and $\{\mathit{bnd}(c,x_1), \mathit{bnd}(c,x_2)\}$ are bounds-equivalent and endpoint-relevant.*

PROOF. Let $D_1 = \mathit{solv}(\{\mathit{dom}(c,x_1), \mathit{dom}(c,x_2)\}, D)$ be the result of domain propagation and $D_2 = \mathit{solv}(\{\mathit{bnd}(c,x_1), \mathit{bnd}(c,x_2)\}, D)$ the result of bounds propagation. First by definition $D_1 = \mathit{iter}(\{\mathit{dom}(c,x_1), \mathit{dom}(c,x_2)\}, D)$. Hence

$$d_1 \in D_1(x_1) \;\text{ iff }\; \sqrt{d_1} \in D(x_2) \tag{3}$$
$$d_2 \in D_1(x_2) \;\text{ iff }\; d_2 \times d_2 \in D(x_1) \tag{4}$$

Clearly also $\inf_{D_1} x_1 = \inf_{D_1} x_2 \times \inf_{D_1} x_2$ and $\sup_{D_1} x_1 = \sup_{D_1} x_2 \times \sup_{D_1} x_2$. By the definition of bounds propagation we have that $\inf_{D_2} x_1 = \inf_{D_2} x_2 \times \inf_{D_2} x_2$ and $\sup_{D_2} x_1 = \sup_{D_2} x_2 \times \sup_{D_2} x_2$. This shows that both sets of propagators are endpoint-relevant.

Now because the endpoints match the conditions of (3) and (4) we have that $\{\inf_{D_2} x_1, \sup_{D_2} x_1\} \subseteq D_1(x)$ and similarly for x_2.

Let $D_2^0 = D$, $D_2^{2i+1} = \mathit{iter}(\mathit{bnd}(c,x_1), D_2^{2i})$, and $D_2^{2i+2} = \mathit{iter}(\mathit{bnd}(c,x_2), D_2^{2i+1})$ for $i \geq 0$. We show by induction that $\inf_{D_2^k} x_j \leq \inf_{D_1} x_j$ and $\sup_{D_2^k} x_j \geq \sup_{D_1} x_j$ for $j = 1,2, k \geq 0$. The base case is straightforward. Suppose $D_2^{k+1}(x_j) \neq D_2^k(x_j)$. We show that the result still holds for D_2^{k+1}. We consider the case when the lower bound of x_1 changes, the other cases are similar. The new lower bound is $\inf_{D_2^{k+1}} x_1 = \lceil \inf_{D_k^2} x_2 \times \inf_{D_k^2} x_2 \rceil$. Now by induction $\inf_{D_2^k} x_2 \leq \inf_{D_1} x_2$, hence $\inf_{D_2^{k+1}} x_1 \leq \inf_{D_1} x_1$.

Finally there exists $k > 0$ such that $D_2^k = D_2$ by the definition of D_2. □

The above results for endpoint-relevance and bounds-equivalence extend straightforwardly to any two variable primitive constraint describing a continuous bijection (over its co-domain), for example $x_1 = x_2 \times x_2 \times x_2$, $x_1 = a \times x_2 \times x_2 \wedge x_2 \geq 0$, and $x_1 = -x_2^4 - x_2^3 - x_2^2 - x_2 - 1 \wedge x_2 \geq 1$. Three variable constraints are in general not endpoint-relevant.

EXAMPLE 3.17. The domain propagators for the linear equation $x_1 = 3x_2 + 5x_3$ from Example 2.2 are not endpoint-relevant. The solutions $\theta_1 = \{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 0\}$ and $\theta_2 = \{x_1 \mapsto 5, x_2 \mapsto 0, x_3 \mapsto 1\}$ illustrate the non-endpoint relevance.

Note that even the domain propagators for $x_1 + x_2 = x_3$ are not endpoint-relevant. Consider the domain-consistent domain $D(x_1) = \{3,5,7,8\}$, $D(x_2) = \{4,12,15\}$, $D(x_3) = \{9,11,15,20\}$ and the bounds-equal $D'(x_1) = \{3,8\}$, $D'(x_2) = \{4,15\}$, $D'(x_3) = \{9,20\}$ which is not domain-consistent. □

The importance of endpoint-relevance is that we can show that conjoining sets of bounds-equivalent and endpoint-relevant propagators maintains these properties.

THEOREM 3.18. *If F_1 and F_2 are bounds-equivalent and endpoint-relevant and F_1' and F_2' are bounds-equivalent and endpoint-relevant, then $F_1 \cup F_1'$ is bounds-equivalent to $F_2 \cup F_2'$ and both $F_1 \cup F_1'$ and $F_2 \cup F_2'$ are endpoint-relevant.*

PROOF. The proof that $F_j \cup F_j'$ is endpoint-relevant given both F_j and F_j' are endpoint-relevant is straightforward.

We construct a series of bounds-equivalent domains beginning from an arbitrary domain D.

Define $D_1^0 = D$, $D_2^0 = D$ and $D_3^0 = D$. Define $D_j^{2k+1} = solv(F_j, D_3^{2k})$ for $j = 1, 2$ and $k \geq 0$. Define $D_4^i = range(D_1^i)$ to be the range domain such that $D_4^i \stackrel{b}{\equiv} D_1^i$ and let $D_3^i = D_4^i \sqcap D$ for $i \geq 0$. Define $D_j^{2k} = solv(F_j', D_3^{2k-1})$ for $j = 1, 2$ and $k > 0$.

We show that $D_1^i \stackrel{b}{\equiv} D_2^i \stackrel{b}{\equiv} D_3^i$ for $i \geq 0$. The base case is trivial.

Now since F_1 and F_2 are bounds-equivalent then $D_1^{2k+1} \stackrel{b}{\equiv} D_2^{2k+1}$ and clearly $D_4^{2k+1} \stackrel{b}{\equiv} D_1^{2k+1}$. By definition $D_1^{2k+1} \sqsubseteq D_3^{2k} \sqsubseteq D$ hence the endpoints of D_1^{2k+1} are in D. Thus $D_3^{2k+1} \stackrel{b}{\equiv} D_4^{2k+1} \stackrel{b}{\equiv} D_1^{2k+1}$.

Similarly since F_1' and F_2' are bounds-equivalent the result holds for $i = 2k, k > 0$.

We must finally reach an i such that $D_3^{i+1} = D_3^i$. Let $D^* = D_3^i$. Clearly then $solv(F_1 \cup F_1', D^*) \stackrel{b}{\equiv} D^*$ since both F_1 and F_1' are endpoint-relevant. Similarly $solv(F_2 \cup F_2', D^*) \stackrel{b}{\equiv} D^*$.

It remains to show that $solv(F_j \cup F_j', D) \stackrel{b}{\equiv} D^*, j = 1, 2$. Clearly since $D^* \sqsubseteq D$ we have that $D^* \stackrel{b}{\equiv} solv(F_j \cup F_j', D^*) \sqsubseteq solv(F_j \cup F_j', D)$ by the monotonicity of $solv$. We now prove that $solv(F_j \cup F_j', D) \sqsubseteq D^*$

We consider the case for $j = 1$, the case for $j = 2$ is identical. We now consider the sequence D_5^i defined as follows: $D_5^0 = D$, $D_5^{2k+1} = solv(F_1, D_5^{2k}), k \geq 0$, and $D_5^{2k} = solv(F_1', D_5^{2k-1}), k > 0$. We show that $D_5^i \sqsubseteq D_3^i, i \geq 0$. The base case is obvious. Clearly $D_5^{2k+1} = solv(F_1, D_5^{2k}) \sqsubseteq solv(F_1, D_3^{2k}) = D_1^{2k+1}$ since $solv$ is monotonic. Now by definition $D_1^{2k+1} \sqsubseteq D_3^{2k+1}$, hence the induction hypothesis holds. The same reasoning applies to D_5^{2k} and D_3^{2k} for $k > 0$. Now there exists i such that $D_5^i = solv(F_j \cup F_j', D) \sqsubseteq D_3^i = D^*$. □

EXAMPLE 3.19. We can now prove that domain propagation or bounds propagation on the example in the introduction

$$[x_1, x_2, x_3, x_4] :: [0..10], x_1 \leq x_2, 2x_2 = 3x_3 + 1, x_3 \leq x_4$$

is bounds-equivalent. The domain and bounds propagators for $2x_2 = 3x_3 + 1$ are bounds-equivalent and endpoint-relevant by Lemma 3.14, and each of the propagators for $x_1 \leq x_2$ and $x_3 \leq x_4$ are endpoint-relevant and equivalent (domain versus bounds). Hence the conjunction is also bounds-equivalent and endpoint-relevant by Theorem 3.18. □

Typically domain propagation is not ever used for linear equations with more than two variables. This results from the fact that finding the solutions to a linear integer equation is NP-hard. Under the assumption that all linear equations involving more than two variables are handled using bounds propagation, we already have enough to show a somewhat surprising result. Using domain propagation (modulo the above discussion) or bounds propagation on linear integer constraints is bounds-equivalent.

PROPOSITION 3.20. *Let C be a conjunction of linear integer constraints excluding linear equations with three or more variables. Let C' be a conjunction of linear equations with three or more variables.*

Then $\{dom(c, x) \mid c \in C, x \in vars(c)\} \cup \{bnd(c, x) \mid c \in C', x \in vars(c)\}$ *is bounds-equivalent to* $\{bnd(c, x) \mid c \in C \cup C', x \in vars(c)\}$. □

3.3 Range-Equivalence

This section discusses how we can go beyond endpoint-relevance.

EXAMPLE 3.21. Consider this variation of the example from the introduction

$$[x_1, x_2, x_3, x_4] :: [0..10], x_1 \leq x_2, x_2 + x_3 = x_4, x_3 \leq x_4$$

The domain and bounds propagators for the constraint $x_2 + x_3 = x_4$ are neither endpoint-relevant nor bounds-equivalent. Yet clearly the only constraint that can generate holes in the domains is $x_2 + x_3 = x_4$. But these holes in the domains are irrelevant to the other constraints. Hence domain propagation or bounds propagation for this constraint should be equivalent.

Similarly if we added the constraint $x_1 \neq 3$, then although it generates a hole in the domain of x_1, this is irrelevant to the constraint $x_2 + x_3 = x_4$. Again we should be able to use bounds propagation rather than domain propagation. □

Hence we introduce the notion of range-equivalence, which ensures that the sets of propagators give bounds-equivalent results for range domains.

DEFINITION 3.22. Two sets of propagators F_1 and F_2 are *range-equivalent*, iff for each range domain D, $solv(F_1, D) \stackrel{b}{\equiv} solv(F_2, D)$.

Clearly range-equivalent propagators detect failure at the same time for input range domains.

PROPOSITION 3.23. *Let F_1 and F_2 be two range-equivalent sets of propagators. For any range domain D, $solv(F_1, D)$ is a false domain, iff $solv(F_2, D)$ is a false domain.* □

We will be interested in determining sets of range-equivalent propagators.

LEMMA 3.24. *Let c be a linear equation $\sum_{i=1}^{n} a_i x_i = d$ with $|a_i| = 1$ for $1 \leq i \leq n$. Then $\{dom(c, x_i)\}$ and $\{bnd(c, x_i)\}$ are bounds-preserving and range-equivalent for $1 \leq i \leq n$.*

PROOF. Assume w.l.o.g. that $a_j = 1$. Let D be a range domain where $[l \mathinner{..} u] = bnd(c, x_j)(D)$. By definition

$$l = d - \sum_{i=1, i \neq j}^{n} \sup_D(a_i x_i)$$

$$u = d - \sum_{i=1, i \neq j}^{n} \inf_D(a_i x_i)$$

We show that for each $d_j \in [l \mathrel{..} u]$ there is a solution $\theta \in D$ of $\sum_{i=1,i\neq j}^{n} a_i x_i = d - d_j$. This proves that $dom(c, x_j)$ is bounds-preserving, and that $dom(c, x_j)(D) = bnd(c, x_j)(D)$

Clearly $\theta_1 = \{x_i \mapsto \sup_D(a_i x_i)\}$ is a solution when $d_j = l$, and $\theta_2 = \{x_i \mapsto \inf_D(a_i x_i)\}$ is a solution when $d_j = u$ by their definition. Now

$$\begin{aligned} u - l &= \theta_2(d - \sum_{i=1,i\neq j}^{n} a_i x_i) - \theta_1(d - \sum_{i=1,i\neq j}^{n} a_i x_i - d) \\ &= \sum_{i=1,i\neq j}^{n} (\sup_D(a_i x_i) - \inf_D(a_i x_i)) \end{aligned}$$

Take d_j such that $l < d_j < u$. Then $u - d_j < u - l$ and hence there exist $e_i \leq \sup_D(a_i x_i) - \inf_D(a_i x_i)$ such that $u - d_j = \sum_{i=1,i\neq j}^{n} e_i$. Now $e_i + \inf_D(a_i x_i) \in D(x_i)$, since $D(x_i)$ is a range and $\theta = \{x_i \mapsto e_i + \inf_D(a_i x_i)\}$ is a solution of $\sum_{i=1,i\neq j}^{n} a_i x_i = d - d_j$ by construction. □

Clearly Examples 2.2 and 2.3 illustrate that domain and bounds propagators for linear integer equations with arbitrary coefficients are not range-equivalent.

In contrast, minimum constraints are range-equivalent.

Lemma 3.25. *Let c be $x_1 = \mathtt{min}(x_2, x_3)$. Then $\{dom(c, x_i) \mid 1 \leq i \leq 3\}$ and $\{bnd(c, x_i) \mid 1 \leq i \leq 3\}$ are bounds-preserving and range-equivalent.*

Proof. Given a range domain D, let $[l_i \mathrel{..} u_i] = bnd(c, x_i)(D) \cap D(x_i)$ for $1 \leq i \leq 3$. We show that there exist solutions $\theta \in D$ for c such that $\theta(x_i) = d_i$ for each value $l_i \leq d_i \leq u_i$ for $1 \leq i \leq 3$. This proves that $dom(c, x_i)$ is bounds-preserving, and that $dom(c, x_i)(D) = bnd(c, x_i)(D) \cap D(x_i)$

Clearly $\{x_1 \mapsto l_1, x_2 \mapsto l_2, x_3 \mapsto l_3\}$ is a solution of c, since $l_1 = l_2$ or $l_1 = l_3$ by definition of the propagators.

Suppose w.l.o.g. $l_1 = l_2$, then $l_3 \geq l_1$ and hence each valuation $\{x_1 \mapsto l_1, x_2 \mapsto l_2, x_3 \mapsto d\}$ for $l_3 \leq d \leq u_3$ is also a solution of c. This gives a solution for each value of x_3.

Now $u_1 \leq \inf\{u_2, u_3\}$, hence $\{x_1 \mapsto d, x_2 \mapsto d, x_3 \mapsto u_3\}$ is a solution of c for $l_1 \leq d \leq u_1$. That gives a solution for each value of x_1.

If $l_3 \geq u_2$ then $u_2 \leq u_1$ by definition of the propagators and we are finished. Suppose $l_3 < u_2$ then we have not yet provided a solution where x_2 takes values in $[u_1 + 1 \mathrel{..} u_2]$. Clearly $\{x_1 \mapsto l_3, x_2 \mapsto d, x_3 \mapsto l_3\}$ with $u_1 + 1 \leq d \leq u_2$ provides these solutions. □

Note that although minimum constraints are range-equivalent, they are not bounds-equivalent.

Example 3.26. Consider the constraint $x_1 = \mathtt{min}(x_2, x_3)$ and domain $D(x_1) = \{1, 3, 5, 7\}$, $D(x_2) = \{2, 4, 6\}$ and $D(x_3) = \{3, 8\}$.

Domain propagation leads to domain D_1 where $D_1(x_1) = \{3\}$, $D_1(x_2) = \{4, 6\}$ and $D_1(x_3) = \{3\}$, while bounds propagation leads to domain D_2 where $D_2(x_1) = \{3, 5\}$, $D_2(x_2) = \{4, 6\}$ and $D_2(x_3) = \{3, 8\}$.

This is because the constraint is not endpoint-relevant. □

Of more interest is the fact that we can produce efficient range-equivalent propagators for complex constraints like `alldifferent`.

Lemma 3.27. *If c is* `alldifferent`$([x_1, \ldots, x_n])$*, then* $\{dom(c, x_i) \mid 1 \leq i \leq n\}$ *and* $\{bnd(c, x_i) \mid 1 \leq i \leq n\}$ *are range-equivalent.*

Proof. See for example [Puget 1998]. □

Once we have established range-equivalence for primitive constraints, we can extend this to larger sets of constraints using Theorem 3.29 below. There are some side conditions about the interaction of variables that first require definition.

Definition 3.28. A variable x is *bounds-only* w.r.t. a set of propagators F, if each propagator $f \in F$ for variable x is bounds-only.

A set of propagators F is *endpoint-relevant* for variables V, if for all domains D, if $D_1 = solv(F, D)$ and $D_2 \stackrel{b}{\equiv} D_1$ and $D_2 =_{\mathcal{V}-V} D_1$, then $solv(F, D_2) \stackrel{b}{\equiv} D_1$.

Theorem 3.29. *Let F_1 and F_2 be range-equivalent. Let F'_1 and F'_2 be range-equivalent. Let F'_1 and F'_2 be endpoint-relevant and bounds-only on variables $V = vars(F_1) \cup vars(F_2)$. Then $F_1 \cup F'_1$ and $F_2 \cup F'_2$ are range-equivalent.*

Proof. We construct a series of bounds-equivalent domains beginning from an arbitrary range domain D.

Define $D^0_1 = D$, $D^0_2 = D$ and $D^0_3 = D$. Define $D^{2k+1}_j = solv(F_j, D^{2k}_3), k \geq 0$, for $j = 1, 2$. Define $D^i_3 = range(D^i_1$, that is the range domain such that $D^i_3 \stackrel{b}{\equiv} D^i_1$. Define $D^{2k}_j = solv(F'_j, D^{2k-1}_3)$, $k > 0$, for $j = 1, 2$.

We show that $D^i_1 \stackrel{b}{\equiv} D^i_2 \stackrel{b}{\equiv} D^i_3, i \geq 0$. The base case is trivial.

Now since F_1 and F_2 are range-equivalent, $D^{2k+1}_1 \stackrel{b}{\equiv} D^{2k+1}_2$ and clearly $D^{2k+1}_3 \stackrel{b}{\equiv} D^{2k+1}_1$.

Similarly since F'_1 and F'_2 are range-equivalent the result holds for $i = 2k, k > 0$.

We must finally reach an i such that $D^{i+1}_3 = D^i_3$. Let $D^* = D^i_3$. Let $E_j = solv(F'_j, D^*)$. Then $E_j =_V D^*$ since F'_j is bounds-only on V and by the definition of D^*. Also $E_1 \stackrel{b}{\equiv} E_2$ since F'_1 and F'_2 are range-equivalent. Let $E'_j = solv(F_j, E_j)$. Clearly $E'_1 \stackrel{b}{\equiv} E'_2$ since $E_1 =_V D^* =_V E_2$ and both F_1 and F_2 only depend on variables in V. And by the definition of D^*, $E'_j =_V D^*$.

In fact $E'_j = solv(F_j \cup F'_j, D)$. Since F'_j is endpoint-relevant and $E'_j =_V D^*$ we have that $solv(F'_j, E'_j) = E'_j$. Clearly E'_j is a fixpoint of $\lambda x.\, solv(F_j \cup F'_j, x \sqcap D)$, for $j = 1, 2$ and hence $E'_j \sqsubseteq solv(F_j \cup F'_j, D)$. It remains to show that $solv(F_j \cup F'_j, D) \sqsubseteq E'_j$.

We consider the case when $j = 1$, the case when $j = 2$ is identical. We now consider the sequence D^i_5 defined as follows: $D^0_5 = D$, $D^{2k+1}_5 = solv(F_1, D^{2k}_5), k \geq 0$, and $D^{2k}_5 = solv(F'_1, D^{2k-1}_5), k > 0$. We show that $D^i_5 \sqsubseteq D^i_3, i \geq 0$. The base case is obvious. Clearly $D^{2k+1}_5 = solv(F_1, D^{2k}_5) \sqsubseteq solv(F_1, D^{2k}_3) = D^{2k+1}_1$ since *solv* is monotonic. Now by definition $D^{2k+1}_1 \sqsubseteq D^{2k+1}_3$, hence the induction hypothesis holds. The same reasoning applies to D^{2k}_5 and D^{2k}_3 for $k > 0$. Now there exists i s.t. $D^i_5 = solv(F_j \cup F'_j, D) \sqsubseteq D^i_3 = D^*$. The final two steps follow from the definition of E'_1. □

Example 3.30. Consider two range-equivalent sets of propagators for the constraint `alldifferent`$([x_1, x_2, x_3])$, one set, F_1, based on domain propagation and the other, F_2, based on bounds propagation.

Let F_1' be domain propagators for $x_1 \leq x_3, 2x_3 + x_4 \leq 6, x_2 + x_5 \leq 4, x_4 = 2x_5 - 1$, and F_2' be bounds propagators for the same system. By Lemma 3.14 and Theorem 3.18 we have that F_1' and F_2' are bounds-equivalent and endpoint-relevant. Clearly on the variables x_1, x_2 and x_3 they are bounds-only.

Consider the initial domain $D(x_i) = [1 .. 3]$ for $1 \leq i \leq 5$. Domain propagation of F_1' obtains $D_1^1(x_1) = [1 .. 2]$, $D_1^1(x_2) = [1 .. 3]$, $D_1^1(x_3) = [1 .. 2]$, $D_1^1(x_4) = \{1, 3\}$, and $D_1^1(x_5) = [1 .. 2]$. Bounds propagation of F_2' obtains the range domain $D_2^1 = \mathit{range}(D_1^1)$. Note that $D_1^1 =_{\{x_1,x_2,x_3\}} D_2^1$.

Domain propagation on F_1 then determines domain D_1^2 which modifies $D_1^2(x_2) = \{3\}$. Similarly for F_2 applied to D_2^1 obtaining D_2^2.

Domain propagation of F_1' now obtains D_1^3 which modifies $D_1^3(x_4) = \{1\}$ and $D_1^3(x_5) = \{1\}$. Similarly for F_2 applied to D_2^2 obtaining D_2^3. Now both fixpoints are reached and $D_2^3 = D_1^3$. □

EXAMPLE 3.31. A well-known program for $SEND + MORE = MONEY$ is:

```
smm(S,E,N,D,M,O,R,Y) :-
     [S,E,N,D,M,O,R,Y] ::  [0..9],
     S >= 1, M >= 1,
                  1000 * S + 100 * E + 10 * N + D
                + 1000 * M + 100 * O + 10 * R + E
     = 10000 * M + 1000 * O + 100 * N + 10 * E + Y,
     alldifferent([S,E,N,D,M,O,R,Y]),
     labelling([S,E,N,D,M,O,R,Y]).
```

Assuming that bounds propagation is used for the large linear equation with more than two variables, then using either bounds or domain propagation for `alldifferent` leads to the same search space being traversed. The result holds using Lemmas 3.7 and 3.27, and Theorem 3.29. □

4. ANALYSING FD PROGRAMS

Now we are ready to devise a bottom-up analysis to discover weaker sets of propagators for a CLP(FD) program that give equivalent search behaviour. We assume we are given a pure CLP(FD) program and must choose for each primitive constraint which implementation by a set of propagators to use.

For simplicity, we only consider pure CLP(FD) programs without data structures. We could extend the approach to CLP(FD) programs with types defined by deterministic finite tree automata using the methodology of [Lagoon and Stuckey 2001].

Note that often constraint programming systems (other than CLP systems) simply build a conjunction of constraints and then apply a predefined search strategy. Since such problems can be described by a CLP(FD) program without data structures, we can apply the same analysis.

Analysis and "optimization" of the CLP(FD) program proceeds in two phases.

Range and endpoint. In the first phase, a bottom-up analysis determines which variables are guaranteed to have a range domain, and which are guaranteed to only appear in endpoint-relevant constraints.

```
labelling([]).
labelling([V|Vs]) :- indomain(V), labelling(Vs).

indomain(V) :- V ≤ inf_D(V).
indomain(V) :- V ≥ inf_D(V) + 1, indomain(V).

labellingff([]).
labellingff(Vs) :- V = choose{v ∈ Vs | |D(v)| is minimal},
    indomain(V),
    Rs = Vs − {V}, labellingff(Rs).

labellingmid([]).
labellingmid([V|Vs]) :- middomain(V), labellingmid(Vs).

middomain(V) :- V = median D(V).
middomain(V) :- V ≠ median D(V), middomain(V).
```

Fig. 1. Pseudo-code implementation of labelling.

Calling context. In the next phase, we determine the calling context (in terms of range and endpoint information) for each literal, and replace it by the appropriate propagation method.

We assume the reader is somewhat familiar with abstract interpretation of CLP programs (see e.g. [García de la Banda et al. 1996; Marriott and Søndergaard 1990]). However, we give self-contained algorithms that make the analysis process accessible to the reader not so familiar with abstract interpretation. We begin by formally defining CLP(FD) programs, and then define the analysis and transformation required for replacing domain propagators by bounds propagators.

4.1 CLP(FD) Programs

We briefly define CLP(FD) programs, for more information see e.g. [Marriott and Stuckey 1998; Van Hentenryck 1989].

An *atom* is of the form $p(x_1, \ldots, x_n)$ where p is a *predicate symbol* and $x_1, \ldots, x_n$ are distinct variables in $\mathcal{V}$. A *literal* is an atom, labelling literal, or primitive constraint. A *goal* is a sequence of literals. A *CLP(FD) program* is a set of rules A `:-` G where A is an atom, and G is a goal.

Note that we assume here a restricted form of programs as all atoms appear with distinct variable arguments. It is easy to translate any program to an equivalent program of this form.

A *labelling literal* is (for our purposes) one of

`labelling`$([x_1, \ldots, x_n])$
`labellingff`$([x_1, \ldots, x_n])$
`labellingmid`$([x_1, \ldots, x_n])$

where $x_1, \ldots, x_n$ are distinct variables. The role of a labelling literal is to ensure that every variable involved eventually takes a fixed value.

There are many kinds of labelling possible, but the three we use `labelling` (default), `labellingff` (first-fail labelling) and `labellingmid` (middle-out value ordering) illustrate the three different kinds of propagation behaviour. `labelling`, and other labellings (such as labelling the variable with least minimum value) only depend on the endpoints of a domain and only add inequality constraints. `labellingff` calculates which variable x to label using the size of the domain $|D(x)|$, hence it depends on the entire domain, but it only adds inequality constraints. `labellingmid` and other labellings not only depend on the entire domain but also add disequality constraints in the disjunction. Pseudo-code for each kind of labelling is given in Figure 1.

The execution of a CLP(FD) program will rely on a mapping from primitive constraints to propagators implementing them. An *implementation* $imp(c)$ of a constraint c is a set of propagators f such that f is correct for c. The *domain-implementation* of c is defined as

$$imp(c) = \{dom(c, x) \mid x \in vars(c)\}$$

while the *bounds-implementation* of c is defined as

$$imp(c) = \{bnd(c, x) \mid x \in vars(c)\}$$

Other kinds of implementation are possible.

The operational semantics of CLP(FD) programs is defined as usual but here we separate the domain from propagators, and restrict to matching data structures (lists) arising in labelling predicates.

A *state* is a triple $\langle\, G \mid F \mid D \,\rangle$ of a goal G, a set of propagators F, and a domain D. A *derivation step* from state $\langle\, G_0 \mid F_0 \mid D_0 \,\rangle$ to state $\langle\, G_1 \mid F_1 \mid D_1 \,\rangle$ in program P, written $\langle\, G_0 \mid F_0 \mid D_0 \,\rangle \Rightarrow_P \langle\, G_1 \mid F_1 \mid D_1 \,\rangle$ is defined as follows. Let $G_0 \equiv L_1, \ldots, L_n$.

—if L_1 is a primitive constraint c, then $F_1 = F_0 \cup imp(c)$, $D_1 = solv(F_1, D_0)$ and
  —if D_1 is a false domain, $G_1 = \Box$ (the empty goal);
  —otherwise $G_1 = L_2, \ldots, L_n$.

—if L_1 is an atom $p(t_1, \ldots, t_m)$ there is a rule $p(s_1, \ldots, s_m)$ `:-` G in the program, and ρ is a renaming such that $\rho(s_i) = t_i$, then $F_1 = F_0$, $D_1 = D_0$, and $G_1 = \rho(G), L_2, \ldots, L_n$.

—if L_1 is a labelling literal $p(t_1, \ldots, t_m)$ there is a rule $p(s_1, \ldots, s_m)$ `:-` G in the definition of the labelling literal, and θ is a substitution such that $\theta(s_i) = t_i$, then $F_1 = F_0$, $D_1 = D_0$, and $G_1 = \theta(G), L_2, \ldots, L_n$.

A *derivation* in P, $S_0 \Rightarrow_P S_1 \Rightarrow_P \cdots \Rightarrow_P S_n$ is a sequence of derivation steps. A derivation is *successful* if $S_n = \langle\, \Box \mid F_n \mid D_n \,\rangle$ and D_n is not a false domain. A derivation is *failed* if $S_n = \langle\, \Box \mid F_n \mid D_n \,\rangle$ and D_n is a false domain. A derivation for a goal G is a derivation for the state $\langle\, G \mid \emptyset \mid D_{init} \,\rangle$ where D_{init} is some default initial domain mapping each variable to some large initial range, for example $D_{init}(x) = [-10^6 \,..\, 10^6]$.

Example 4.1. A sample successful derivation for the goal

$$x_1 \geq 0, x_2 \leq 4, 2x_1 = x_2, \texttt{labelling}([x_1, x_2])$$

using the domain-implementation for constraints is shown below. Only constraints collected are shown rather than the individual domain propagators.

$$
\begin{array}{ll}
 & \langle\, x_1 \geq 0, x_2 \leq 4, 2x_1 = x_2, \texttt{labelling}([x_1,x_2]) \,\|\, \emptyset \,\|\, D_{init} \,\rangle \\
\Rightarrow_P & \langle\, x_2 \leq 4, 2x_1 = x_2, \texttt{labelling}([x_1,x_2]) \,\|\, x_1 \geq 0 \,\|\, D_1 \,\rangle \\
\Rightarrow_P & \langle\, 2x_1 = x_2, \texttt{labelling}([x_1,x_2]) \,\|\, x_1 \geq 0 \wedge x_2 \leq 4 \,\|\, D_2 \,\rangle \\
\Rightarrow_P & \langle\, \texttt{labelling}([x_1,x_2]) \,\|\, x_1 \geq 0 \wedge x_2 \leq 4 \wedge 2x_1 = x_2 \,\|\, D_3 \,\rangle \\
\Rightarrow_P & \langle\, \texttt{indomain}(x_1), \texttt{labelling}([x_2]) \,\|\, x_1 \geq 0 \wedge x_2 \leq 4 \wedge 2x_1 = x_2 \,\|\, D_3 \,\rangle \\
\Rightarrow_P & \langle\, x_1 \leq 0, \texttt{labelling}([x_2]) \,\|\, x_1 \geq 0 \wedge x_2 \leq 4 \wedge 2x_1 = x_2 \,\|\, D_3 \,\rangle \\
\Rightarrow_P & \langle\, \texttt{labelling}([x_2]) \,\|\, x_1 \geq 0 \wedge x_2 \leq 4 \wedge 2x_1 = x_2 \wedge x_1 \leq 0 \,\|\, D_4 \,\rangle \\
\Rightarrow_P & \langle\, \texttt{indomain}(x_2), \texttt{labelling}([]) \,\|\, x_1 \geq 0 \wedge x_2 \leq 4 \wedge 2x_1 = x_2 \wedge x_1 \leq 0 \,\|\, D_4 \,\rangle \\
\Rightarrow_P & \langle\, x_2 \leq 0, \texttt{labelling}([]) \,\|\, x_1 \geq 0 \wedge x_2 \leq 4 \wedge 2x_1 = x_2 \wedge x_1 \leq 0 \,\|\, D_4 \,\rangle \\
\Rightarrow_P & \langle\, \texttt{labelling}([]) \,\|\, x_1 \geq 0 \wedge x_2 \leq 4 \wedge 2x_1 = x_2 \wedge x_1 \leq 0 \wedge x_2 \leq 0 \,\|\, D_4 \,\rangle \\
\Rightarrow_P & \langle\, \square \,\|\, x_1 \geq 0 \wedge x_2 \leq 4 \wedge 2x_1 = x_2 \wedge x_1 \leq 0 \wedge x_2 \leq 0 \,\|\, D_4 \,\rangle
\end{array}
$$

where

D	$D(x_1)$	$D(x_2)$
D_1	$[0 \mathrel{..} 10^6]$	$[-10^6 \mathrel{..} 10^6]$
D_2	$[0 \mathrel{..} 10^6]$	$[-10^6 \mathrel{..} 4]$
D_3	$[0 \mathrel{..} 2]$	$\{0, 2, 4\}$
D_4	$\{0\}$	$\{0\}$

□

4.2 Range and Endpoint Descriptions

The first phase is a simple bottom-up abstract interpretation where we determine which variables must have range domains, and which variables are only involved in endpoint-relevant constraints.

For simplicity we start with the simple case of a single conjunction of constraints. Essentially we examine each constraint to determine two classes of variables. The variables for which the constraint may cause holes in their domains become *non-range* variables. Each variable for which the constraint is not endpoint-relevant becomes a *non-endpoint-relevant* variable. We conjoin these sets of variables to get a total set of non-range and non-endpoint-relevant variables. The variables not in these sets are guaranteed to, always have a range domain, and, respectively, to not be involved in non-endpoint-relevant constraints.

Example 4.2. The (non-)range description of $x_1 \leq x_2$ is $\{\}$ since each of the variables appearing in it is guaranteed to have a range domain. The (non-)range description of $2x_2 + 5x_3 = 4$ is $\{x_2, x_3\}$ indicating that both x_2 and x_3 may not have range domains. Its (non-)endpoint description is $\{\}$ indicating that $\{x_2, x_3\}$ only appear in this endpoint-relevant constraint. The (non-)range and (non-)endpoint descriptions for $x_5 = x_6 \times x_6$ are both $\{x_5, x_6\}$ indicating that they may not be guaranteed to have a range domain, and that this is not an endpoint-relevant constraint.

Consider the conjunction of constraints

$$x_1 \le x_2, x_3 \ne 4, 2x_2 + 5x_3 = 4, x_3 \le x_4 + x_5, x_5 = x_6 \times x_6$$

Then the variables which are made non-range are $\{x_3\}$ from $x_3 \ne 4$, $\{x_2, x_3\}$ from $2x_2 + x_3 = 4$ and $\{x_5, x_6\}$ from $x_5 = x_6 \times x_6$. In total the (non-)range variables are $\{x_2, x_3, x_5, x_6\}$. Clearly x_1 and x_4 are guaranteed to have range domains.

The variables which are made (non-)endpoint are $\{x_5, x_6\}$ from $x_5 = x_6 \times x_6$. Each constraint involving variables $\{x_1, x_2, x_3, x_4\}$ is endpoint-relevant. □

We will apply this idea to analysis of CLP(FD) programs using a slight extension. Rather than collecting sets of non-range and non-endpoint-relevant variables we use Boolean formulae to define the non-range and non-endpoint relevant variables. This allows us to express bounds-preserving constraints more accurately.

The bottom-up analysis determines for each user-defined constraint $p(x_1, \ldots, x_n)$ two Boolean formulae[1] describing its (non-)range and (non-)endpoint behaviour. The intuition is that the Boolean variable corresponding to a variable x is true if the variable is not guaranteed to have a range domain (resp. not guaranteed to only appear in endpoint-relevant constraints).

Example 4.3. The (non-)range description of $x_1 \le x_2$ is true since each of the variables appearing in it is guaranteed to have a range domain. The (non-)range description of $2x_2 + 5x_3 = 4$ is $x_2 \wedge x_3$ indicating that both x_2 and x_3 may not have range domains. Its (non-)endpoint description is true indicating that $\{x_2, x_3\}$ only appear in this endpoint-relevant constraint. The (non-)range and (non-)endpoint descriptions for $x_5 = x_6 \times x_6$ are both $x_5 \wedge x_6$ indicating that they may not be guaranteed to have a range domain, and that this is not an endpoint-relevant constraint.

Consider the conjunction of constraints

$$x_1 \le x_2, x_3 \ne 4, 2x_2 + 5x_3 = 4, x_3 \le x_4 + x_5, x_5 = x_6 \times x_6$$

Then the (non-)range description is the conjunction of the (non-)range descriptions of the individual constraints: x_3 from $x_3 \ne 4$, $x_2 \wedge x_3$ from $2x_2 + x_3 = 4$ and $x_5 \wedge x_6$ from $x_5 = x_6 \times x_6$. In total the (non-)range variables are $x_2 \wedge x_3 \wedge x_5 \wedge x_6$.

Similarly the (non-)endpoint description is the conjunction of the (non)-endpoint descriptions of the individual constraints: that is $x_5 \wedge x_6$. □

Example 4.4. To see an example of where we gain advantages from the Boolean representation consider

$$x_1 \le x_2, x_2 + x_3 + x_4 = 3, x_4 - x_3 \le x_5, x_5 \ne 3$$

Then the (non-)range description of $x_2 + x_3 + x_4 = 3$ is $(x_2 \leftrightarrow x_3) \wedge (x_2 \leftrightarrow x_4)$ indicating that it is bounds-preserving. The total (non-)range description is $(x_2 \leftrightarrow x_3) \wedge (x_2 \leftrightarrow x_4) \wedge x_5$ which we will interpret to mean that only x_5 may not have a range domain. □

Definition 4.5. The abstract domain A for both descriptions used is a simple (inverted) domain of Boolean formulae defined as follows:

[1] For most of the primitive constraints we could simply restrict ourselves to conjunctions of positive literals (i.e. sets of Boolean variables). We use the Boolean domain for generality.

$$\phi_1 \sqsubseteq_A \phi_2 \text{ iff } \phi_2 \rightarrow \phi_1. \quad \bot_A = \text{true}, \top_A = \text{false}.$$
$$\phi_1 \sqcap_A \phi_2 = \phi_1 \vee \phi_2. \quad \phi_1 \sqcup_A \phi_2 = \phi_1 \wedge \phi_2.$$

DEFINITION 4.6. The meaning of a range description ϕ is defined by the concretization function γ_R defined below. We first introduce some auxiliary notation. Define $true(\phi, c)$ as the formula $(\exists(\mathcal{V} - vars(c)\phi) \leftrightarrow \text{true}$. This formula holds whenever ϕ projected onto the variables of c is equivalent to true. Similarly define $iff(\phi, c)$ as the formula $(\exists(\mathcal{V} - vars(c)\phi) \leftrightarrow (\wedge_{x,y \in vars(c)} x \leftrightarrow y)$. This formula holds when ϕ projected onto the variables of c gives a formula where all variables of c are equivalent.

$$\gamma_R(\phi) = \{C \mid C \text{ satisfies (1) and (2)}\}$$

where

(1) $\forall c \in C$ where $true(\phi, c), \forall x \in vars(c)\ dom(c, x)$ is bounds-only
(2) $\forall c \in C$ where $iff(\phi, c), \forall x \in vars(c)\ dom(c, x)$ is bounds-preserving

The meaning of an endpoint description ϕ is defined by the concretization function γ_E:

$$\gamma_E(\psi) = \{C \mid \forall c \in C \text{ where } true(\phi, c), \{dom(c, y) \mid y \in vars(c)\} \text{ endpoint-relevant}\}$$

We can define the approximation function α for the range and endpoint descriptions for each primitive constraint as in Table I. Here we treat `labelling` goals, which drive the search for solution, as primitive constraints since their implementation involves data-structure manipulation.

Note that the only interesting Boolean formulae arise from range descriptions for linear equations with unit coefficients and min constraints since they are bounds-preserving.

We can lift the analysis to conjunctions of constraints simply by conjoining the descriptions, so abstract conjunction is defined as $\mathsf{Aconj}_R(\phi, \phi') = \mathsf{Aconj}_E(\phi, \phi') = \phi \wedge \phi'$. We can similarly (inaccurately) handle disjunctions by conjunction. So abstract disjunction is defined as $\mathsf{Adisj}_R(\phi, \phi') = \mathsf{Adisj}_E(\phi, \phi') = \phi \wedge \phi'$. Projection of descriptions onto a set of variables V is Boolean existential quantification, $\mathsf{Aproj}_R(V, \phi) = \mathsf{Aproj}_E(V, \phi) = \exists(\mathcal{V} - V)\phi$.

For recursive programs we can find the least fixpoint in the usual manner (see e.g. [Marriott and Søndergaard 1990]). Note that since there are no infinite ascending chains this process is finite. We can alternatively use a constraint based fixpoint rule (as in Hindley-Milner type inference, see e.g. [Demoen et al. 1999]) which simply ensures that recursive calls have the same descriptions as the head. This is more inaccurate but sound. The function $\mathsf{analyse}_A(G)$ shown in Figure 2 analyses a goal G using abstract domain A (one of R or E). The function below will not terminate on recursive programs, but it is straightforward to extend it to do so, by recognizing recursive calls and calculating fixpoints appropriately. Hence $\mathsf{analyse}_R(G)$ and $\mathsf{analyse}_E(G)$ return the range and endpoint descriptions for a goal G.

It is straightforward to show that the analysis is correct.

THEOREM 4.7. *If C is a constraint arising in a derivation for G then $C \in \gamma_R(\mathsf{analyse}_R(G))$, and $C \in \gamma_E(\mathsf{analyse}_E(G))$.*

Table I. Range and endpoint descriptions for primitive constraints.

Constraint	α_R	α_E
true	true	true
$\sum_{i=1}^n a_i x_i \leq d$	true	true
$x_1 = d$	true	true
$a_1x_1 + a_2x_2 = d$, $\lvert a_i \rvert = 1$	$x_1 \leftrightarrow x_2$	true
$a_1x_1 + a_2x_2 = d$	$x_1 \wedge x_2$	true
$\sum_{i=1}^n a_i x_i = d, n > 2$, see[2]	$\wedge_{i=1}^n x_i$	$\wedge_{i=1}^n x_i$
$\sum_{i=1}^n a_i x_i = d$, $n > 2$, $\lvert a_i \rvert = 1$	$\wedge_{i=2}^n (x_1 \leftrightarrow x_i)$	$\wedge_{i=1}^n x_i$
$\sum_{i=1}^n a_i x_i \neq d$	$\wedge_{i=1}^n x_i$	true
$x_0 \Leftrightarrow \sum_{i=1}^n a_i x_i \leq d$	true	true
$x_0 \Leftrightarrow \sum_{i=1}^n a_i x_i = d$	$\wedge_{i=1}^n x_i$	$x_0 \wedge \wedge_{i=1}^n x_i$
$x_0 \Leftrightarrow \sum_{i=1}^n a_i x_i \neq d$	$\wedge_{i=1}^n x_i$	$x_0 \wedge \wedge_{i=1}^n x_i$
$x_1 = \neg x_2$	true	true
$x_1 = (x_2 \mathbin{\&\&} x_3)$	true	true
$x_1 = (x_2 \mathbin{\Vert} x_3)$	true	true
$x_1 = (x_2 \Rightarrow x_3)$	true	true
$x_1 = (x_2 \Leftrightarrow x_3)$	true	true
$x_1 = x_2 \times x_3$	$x_1 \wedge x_2 \wedge x_3$	$x_1 \wedge x_2 \wedge x_3$
$x_1 = x_2 \times x_2 \wedge x_2 \geq 0$	$x_1 \wedge x_2$	true
$x_1 = x_2 \times x_2$	$x_1 \wedge x_2$	$x_1 \wedge x_2$
$x_1 = \lvert x_2 \rvert$	$x_1 \wedge x_2$	$x_1 \wedge x_2$
$x_1 = \texttt{min}(x_2, x_3)$	$(x_1 \leftrightarrow x_2) \wedge (x_1 \leftrightarrow x_3)$	$x_1 \wedge x_2 \wedge x_3$
`alldifferent`$([x_1, \ldots, x_n])$	$\wedge_{i=1}^n x_i$	$\wedge_{i=1}^n x_i$
`default`$([x_1, \ldots, x_n])$	$\wedge_{i=1}^n x_i$	$\wedge_{i=1}^n x_i$
`labelling`$([x_1, \ldots, x_n])$	true	true
`labellingff`$([x_1, \ldots, x_n])$	true	$\wedge_{i=1}^n x_i$
`labellingmid`$([x_1, \ldots, x_n])$	$\wedge_{i=1}^n x_i$	$\wedge_{i=1}^n x_i$

EXAMPLE 4.8. Consider the following program:

```
g(x1,x2,x3,x4,x5) :- x5 ≠ 6, p(x1,x2,x3,x4,x5).
g(x1,x2,x3,x4,x5) :- x1 = 3x2 + 4x4.
p(x1,x2,x3,x4,x5) :- alldifferent([x1,x2,x3]),
                     q(x1,x2,x3,x4,x5).
q(x1,x2,x3,x4,x5) :- x1 ≤ x6, x6 ≤ x2, 2x3 + x4 ≤ 6,
                     x2 + x5 ≤ 4, x4 = 2x5 − 1.
```

[2]We often *apriori* choose bounds propagation for these constraints in which case the description is (true, true).

```
analyse_A(A_1, ..., A_m)
    AP := true
    for i = 1..m
        case A_i of
        primitive constraint or labelling literal c :
            AP := Aconj_A(AP, α_A(c))
        atom p(y_1, ..., y_n):
            foreach rule p(x_1, ..., x_n) :- B_1, ..., B_k
                let ρ be the renaming {x_i ↦ y_i}
                AP' := analyse_A(B_1, ..., B_k)
                AP'' := Aproj_A({x_1, ..., x_n}, AP')
                AP := Aconj_A(AP, ρ(AP''))
    return AP
```

Fig. 2. Algorithm for simple bottom-up analysis of goal $A_1, \ldots, A_m$ using abstract domain A.

The (range,endpoint) answer descriptions for each literal of the program are shown in the table below:

$x_1 \leq x_6$	$(\text{true}, \text{true})$
$x_6 \leq x_2$	$(\text{true}, \text{true})$
$2x_3 + x_4 \leq 6$	$(\text{true}, \text{true})$
$x_2 + x_5 \leq 4$	$(\text{true}, \text{true})$
$x_4 = 2x_5 - 1$	$(x_4 \wedge x_5, \text{true})$
$x_1 = 3x_2 + 4x_4$	$(x_1 \wedge x_2 \wedge x_4, x_1 \wedge x_2 \wedge x_4)$
$x_5 \neq 6$	(x_5, true)
`q`$(x_1, x_2, x_3, x_4, x_5)$	$(x_4 \wedge x_5, \text{true})$
`alldifferent`$([x_1, x_2, x_3])$	$(x_1 \wedge x_2 \wedge x_3, x_1 \wedge x_2 \wedge x_3)$
`p`$(x_1, x_2, x_3, x_4, x_5)$	$(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5,$ $x_1 \wedge x_2 \wedge x_3)$
`g`$(x_1, x_2, x_3, x_4, x_5)$	$(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5,$ $x_1 \wedge x_2 \wedge x_3 \wedge x_4)$

□

4.3 Determining Calling Contexts

Unlike many analysis-based optimizations, here we need to understand for each primitive constraint, the effect of the remainder of the program on the variables that it involves. This is crucially important in determining whether domain propagation for the constraint will be different to bounds propagation. Even if a constraint can cause holes in the domain of its variables this may be unimportant, if there are no other constraints involving these variables that act differently if holes are present.

Given a primitive constraint and a description of the Range and Endpoint information from the other constraints upon its variables, we can determine when it is safe to use the bounds propagators for the constraint. Table II gives the weakest allowable description for each component that allows the bounds propagators to be used.[3] Each of these optimizations is justified by a lemma. The exception is `labellingff`, which we can replace by a version which uses the calculation

[3]So false allows any description, while true requires that the description is exactly true.

Table II. Weakest possible descriptions to allow the use of bounds propagators.

Constraint	ϕ_R	ϕ_E	Lemma
$\sum_{i=1}^{n} a_i x_i \leq d$	false	false	3.7
$\sum_{i=1}^{n} a_i x_i \neq d$	false	true	3.13
$a_1 x_1 + a_2 x_2 = d$	false	true	3.14
$\sum_{i=1}^{n} a_i x_i = d$, $n > 2$, $\lvert a_i \rvert = 1$	true	false	3.24
$x_0 \Leftrightarrow \sum_{i=1}^{n} a_i x_i \leq d$	false	false	3.8
$x_1 = \neg x_2$	false	false	3.9
$x_1 = (x_2\ \&\&\ x_3)$	false	false	3.9
$x_1 = (x_2 \mid\mid x_3)$	false	false	3.9
$x_1 = (x_2 \Rightarrow x_3)$	false	false	3.9
$x_1 = (x_2 \Leftrightarrow x_3)$	false	false	3.9
$x_1 = x_2 \times x_2 \wedge x_2 \geq 0$	false	true	3.16
$x_1 = \texttt{min}(x_2, x_3)$	true	false	3.25
$\texttt{alldifferent}([x_1, \ldots, x_n])$	true	true	3.27
$\texttt{labellingff}([x_1, \ldots, x_n])$	true	true	—

$\sup_D x - \inf_D x$ rather than $|D(x)|$ to determine the variable with the smallest domain, if all the variables involved are guaranteed to have range domains.

The calling contexts for each literal in the program are determined using a top-down analysis starting from an initial entry point, say `main`. We can mimic multiple entry points G_1 to G_n by simply defining `main` as

$$\texttt{main :- } G_1. \quad \ldots \quad \texttt{main :- } G_n.$$

The analysis starts from the calling pattern `main` : (true,true).

Given we are processing a calling pattern $p(x_1, \ldots, x_n) : (CP_R, CP_E)$, we process each rule of the form

$$p(x_1, \ldots, x_n) \texttt{ :- } A_1, \ldots, A_m$$

by determining the calling context for each literal A_i as the conjunction of the analysis answers for $A_j, 1 \leq i \neq j \leq m$ with the calling description CP. The algorithm is formalized in Figure 3. Initially it is called with an empty table of previous optimizations ($Table$).

Example 4.9. Returning to the program of Example 4.8 and assuming an entry point $\texttt{g}(x_1, x_2, x_3, x_4, x_5)$, transformation determines calling contexts (ignoring inequalities):

transform$(L : (CP_R, CP_E),\ Table)$
  case L **of**
  primitive constraint or labelling literal c:
    if (CP_R, CP_E) satisfies conditions in Table II
      return $bnd(c)$
    else return $dom(c)$
  atom $p(y_1, \ldots, y_n)$:
    if $\exists L' : (CP'_R, CP'_E) \mapsto L'' \in Table$ and
        renaming ρ such that $\rho(L' : (CP'_R, CP'_E)) = L : (CP_R, CP_E)$
      return $\rho(L'')$
    let p' be a new predicate symbol not in $Table$
    $Table := Table \cup \{p(y_1, \ldots, y_n) : (CP_R, CP_E) \mapsto p'(y_1, \ldots, y_n)\}$
    foreach rule $p(x_1, \ldots, x_n)$ `:-` $A_1, \ldots, A_m$
      let ρ be the renaming $\{x_i \mapsto y_i\}$
      $G :=$ true
      for $i = m..1$
        $CP'_R := \mathsf{Aproj}_R(vars(A_i), \rho(CP_R) \wedge \bigwedge\{\mathsf{analyse}_R(A_j) \mid 1 \leq j \neq i \leq m\})$
        $CP'_E := \mathsf{Aproj}_E(vars(A_i), \rho(CP_E) \wedge \bigwedge\{\mathsf{analyse}_E(A_j) \mid 1 \leq j \neq i \leq m\})$
        $O_i := \mathsf{transform}(A_i : (CP'_R, CP'_E),\ Table)$
        $G := (O_i, G)$
      output $p'(x_1, \ldots, x_n)$ `:-` G
    return $p'(y_1, \ldots, y_n)$

Fig. 3. Algorithm for transforming calling pattern $L : (CP_R, CP_E)$ given previous optimizations in $Table$.

$\mathtt{g}(x_1, x_2, x_3, x_4, x_5)$	:	$(\text{true}, \text{true})$
$\mathtt{p}(x_1, x_2, x_3, x_4, x_5)$	:	(x_5, true)
$\mathtt{alldifferent}([x_1, x_2, x_3])$	:	$(\text{true}, \text{true})$
$\mathtt{q}(x_1, x_2, x_3, x_4, x_5)$	:	$(x_1 \wedge x_2 \wedge x_3 \wedge x_5,$ $x_1 \wedge x_2 \wedge x_3)$
$x_5 \neq 6$	:	(x_5, true)
$x_1 = 3x_2 + 4x_4$	:	$(\text{true}, \text{true})$
$x_4 = 2x_5 - 1$	:	(x_5, true)

Hence we replace the `alldifferent` constraint and the constraints $x_5 \neq 6$ and $x_4 = 2x_5 - 1$ by their bounds propagation versions. The program output by the transformation is

```
g(x1,x2,x3,x4,x5) :- bnd(x5 ≠ 6), p(x1,x2,x3,x4,x5).
g(x1,x2,x3,x4,x5) :- dom(x1 = 3x2 + 4x4).
p(x1,x2,x3,x4,x5) :- bnd(alldifferent([x1,x2,x3])),
                     q(x1,x2,x3,x4,x5).
q(x1,x2,x3,x4,x5) :- bnd(x1 ≤ x6), bnd(x6 ≤ x2),
                     bnd(2x3 + x4 ≤ 6),
                     bnd(x2 + x5 ≤ 4),
                     bnd(x4 = 2x5 - 1).
```

□

Note that the optimization is multi-variant, that is it can produce multiple specialized versions of the same predicate. The result of the transformation is a new program with primitive constraints and labelling literals annotated by which propagator *implementation* should be used for them. We assume that the domain implementation is used for all primitive constraints and labelling literals in the original program. The transformed program has the same search space as the original.

THEOREM 4.10. *If P is a CLP(FD) program and P' is the program output by* $\mathsf{transform}(L : (\text{true}, \text{true}), \emptyset)$ *then for any derivation of L in P,* $\langle L \,\|\, \emptyset \,\|\, D_{init} \rangle \Rightarrow^*_P \langle G_n \,\|\, F_n \,\|\, D_n \rangle$, *there is a corresponding derivation of L in P',* $\langle L \,\|\, \emptyset \,\|\, D_{init} \rangle \Rightarrow^*_{P'} \langle G_n \,\|\, F'_n \,\|\, D'_n \rangle$ *such that F_n and F'_n are range-equivalent and* $D_n \stackrel{b}{\equiv} D'_n$.

PROOF. (Sketch) Clearly the programs P and P' are identical except for the implementation of the constraints (and labelling literals). The implementation replacements made in P' are individually justified by the Lemmas shown in Table II and Theorems 3.18 and 3.29. This ensures that during execution of the programs the conjunctions of propagators collected in F_n and F'_n are range-equivalent. Since $D_n = solv(F_n, D_{init})$ and $D'_n = solv(F'_n, D_{init})$ by the monotonicity of $solv$, then clearly $D_n \stackrel{b}{\equiv} D'_n$. □

One has to be quite careful to go beyond the transformations allowed here, because interaction of propagators can be subtle.

EXAMPLE 4.11. Consider the goal

$$\texttt{alldifferent}([x_1, x_2, x_3, x_4, x_5, x_6]),$$
$$x_6 = x_1 + 3, x_4 = x_1 + 3.$$

Since the equations are bounds-preserving we might assume that the `alldifferent` bounds and domain propagators will be equivalent. This is not the case. Consider the domain $D(x_1) = D(x_3) = [1 .. 3]$, $D(x_2) = \{2\}$, $D(x_4) = D(x_5) = D(x_6) = [4 .. 6]$ then domain propagation and bounds propagation are not the same. E.g. `alldifferent` domain propagator gives $D(x_1) = D(x_3) = \{1, 3\}$, $D(x_2) = \{2\}$, $D(x_4) = D(x_5) = D(x_6) = [4 .. 6]$ but then domain propagation on the equalities gives $D(x_4) = D(x_6) = \{4, 6\}$. Subsequently, the `alldifferent` domain propagator gives $D(x_5) = \{5\}$. The `alldifferent` bounds propagator gives $D(x_1) = D(x_3) = [1 .. 3]$, $D(x_2) = \{2\}$, $D(x_4) = D(x_5) = D(x_6) = [4 .. 6]$ and there is no further propagation. The results are not bounds-equal. □

There are a number of obvious ways to improve this analysis. We can eliminate (non-)range information about variables with initial domain of the form $[l .. l + 1]$ (most notably Boolean variables $[0 .. 1]$) since they always have range domains. We can use a preliminary groundness analysis to determine which variables will always be fixed, and then use this information to treat constraints in simpler forms, e.g. the constraint $x_1 = x_2 \times x_3$ becomes a two variable equation, if x_2 is always fixed by the time the constraint is reached.

5. EXPERIMENTAL EVALUATION

We have constructed a prototype analyser and transformer for pure CLP(FD) programs. Here we give experiments to illustrate the effect of the transformation.

We illustrate the effect of the transformation on four classes of benchmarks. The first class includes NP-hard graph problems and multi-knapsack problems with unit values. The graph examples (for example, see [Garey and Johnson 1979]) are vertex cover (`vc-*`) and independent sets (`is-*`) modeled in the natural way using

Table III. Example programs used.

Program	Nodes	Search
vc-20	125	best
vc-40	6 339	best
is-20	93	best
is-40	1 541	best
mk-1	135 581	best
mk-2	865 163	best
photo-1	10 079	best
photo-2	10 079	best

Program	Nodes	Search
smm	7	all
donald	10 967	all
magic-5	6 821	first
golomb-8	6 489	best
golomb-9	34 909	best
golomb-10	191 049	best
sched-bridge	61	best
sched-orb06	57 107	best
sched-orb09	5 647	best
sched-la18	19 173	best
sched-mt10	43 627	best

Boolean variables indicating which vertices are in the selected set. The graphs used are random graphs of 20 and 40 nodes. The constraints are all inequalities except the objective function which is defined using a large linear equation with unit coefficients. The multi-knapsack problems (`mk-*`) are similar and use integer variables for the number of selected items. The multiple resource restrictions are expressed by linear inequality constraints, the objective function is again a linear equality involving all variables with unit coefficients. Both instances are based on the data set given in [Van Hentenryck 1999, Section 2.1.8]. Analysis shows that we can replace domain propagation on the linear equation by bounds propagation without affecting search space.

The second class includes well-known examples `smm` (see Example 3.31), `donald` ($DONALD + GERALD = ROBERT$), magic squares `magic-5` and Golomb ruler problems (`golomb-*`). Here we assume bounds propagation is used on linear equations with more than three variables. Analysis shows we can use bounds propagation on the single `alldifferent` constraint in each benchmark without affecting search space.

The third class of examples uses a simple placement problem: find the maximal number of satisfied preferences for placing two persons next to each other in a photographic picture. Both `photo-1` and `photo-2` use refied constraints for expressing satisfaction of preferences with a Boolean variable. The total satisfaction then is computed by a large linear equation ranging over these Boolean variables. While `photo-1` uses reified linear equations, `photo-2` uses reified linear inequalities to express preferences. Analysis shows for `photo-1` that bounds propagation can be used on the large linear equation. For `photo-2`, bounds propagation can also used for the single occuring `alldifferent` constraint, since the constraints used for preferences are reified linear inequalities.

The fourth class of examples are scheduling examples: including the well-known bridge scheduling example [Dincbas et al. 1990], the remainder are job-shop scheduling examples taken from J.E. Beasley's OR Library [Beasley]. Here analysis shows that the `cumulativeEF` constraint (using a generalization of edge-finding [Martin and Shmoys 1996]) appearing in the benchmarks only requires bounds propagation.

Table III gives the size of the search space (in searched nodes) and the search

Table IV. Comparison of original and transformed programs for Mozart

	Original			Transformed		
Program	DomChg	Exec	Time	DomChg	Exec	Time
`vc-20`	767	490	15.96	=	=	−70.8%
`vc-40`	83 292	24 616	2 601.63	=	=	−87.4%
`is-20`	271	194	13.83	=	=	−75.6%
`is-40`	9 240	3 477	591.25	=	=	−88.9%
`mk-1`	1 650 536	1 584 948	16 689.40	=	=	−51.4%
`mk-2`	13 290 127	9 240 208	86 143.00	=	=	−49.6%
`smm`	33	26	0.38	=	=	−29.8%
`donald`	22 605	34 910	562.34	−13.5%	+16.3%	−31.6%
`magic-5`	95 910	111 327	857.04	+18.8%	+21.3%	−36.9%
`golomb-8`	254 881	294 816	1 522.21	−3.5%	+2.8%	−33.7%
`golomb-9`	1 861 679	2 146 317	11 687.60	−6.6%	−0.1%	−32.0%
`golomb-10`	13 721 156	15 747 604	91 230.00	−8.2%	−2.2%	−29.3%
`photo-1`	47 538	57 852	1 173.27	=	+0.0%	−14.1%
`photo-2`	52 824	79 814	533.06	−3.2%	+4.5%	−31.5%
`sched-bridge`	3 942	11 973	14.74	=	−0.2%	+37.5%
`sched-orb06`	2 969 983	6 047 760	37 634.50	−5.2%	−1.0%	+37.7%
`sched-orb09`	307 722	623 668	4 107.17	−5.0%	−0.5%	+35.5%
`sched-la18`	386 178	898 733	6 994.25	−4.6%	−0.9%	+33.2%
`sched-mt10`	3 023 437	6 107 850	31 248.30	−7.2%	−2.3%	+39.0%

strategy used for each problem: find all solutions (all in column Search), the first solution (first), or a best solution (best). It should be noted that, of course, the number of nodes searched is exactly the same for both the original and transformed program. This has been empirically checked for all examples and all systems used.

All but the multi-knapsack and the scheduling benchmarks use default labelling `labelling`. The labelling for the multi-knapsack problems split the domains of variables according to the arithmetic mean of infimum and supremum of a variable (and thus are very close to the default labelling). The scheduling benchmarks use a labelling strategy similar to that mentioned in [Baptiste et al. 2001] (the labelling strategy considers and contributes bounds information only and hence is equivalent to `labelling` for the purposes of the analysis).

The numbers have been taken on a standard personal computer with a 1.2 GHz Pentium III processor, 256 MB of main memory, and Windows XP Professional as operating system. All runtimes are given as wall-time as the arithmetic mean of 25 runs, where the coefficient of deviation is always less than 3.8%.

We have used the Mozart implementation of Oz [Mozart Consortium 1999] (version 1.2.5) and the more experimental Gecode system.[4] While both systems are not directly based on CLP(FD), both original and transformed programs execute with the same semantics as CLP(FD) programs. The choice of systems is mainly motivated by the fact that only few available systems implement domain consistent propagators for linear constraints.

[4]Gecode is currently under development by the first author and is available upon request.

Table V. Comparison of original and transformed programs for Gecode

Program	Original DomChg	Original Exec	Original Time	Transformed DomChg	Transformed Exec	Transformed Time
vc-20	909	383	35.77	=	−16.4%	−98.5%
vc-40	exceeds time limit					< −99.9%
is-20	337	204	93.07	=	−27.9%	−99.7%
is-40	exceeds time limit					< −99.9%
mk-1	exceeds time limit					< −99.9%
mk-2	exceeds time limit					< −99.9%
smm	63	19	0.05	−7.9%	=	−40.3%
donald	55 741	24 992	120.67	−1.5%	+0.0%	−50.1%
magic-5	106 717	78 441	301.24	−1.1%	+0.9%	−59.9%
golomb-8	375 194	329 746	909.72	+0.5%	+0.9%	−59.1%
golomb-9	2 772 958	2 420 565	8 430.12	+0.6%	+0.9%	−65.9%
golomb-10	20 641 004	17 934 193	79 546.00	+0.7%	+1.1%	−72.2%
photo-1	72 672	187 844	217.43	=	−15.2%	−35.1%
photo-2	75 470	92 292	122.62	=	−8.5%	−38.9%

Table IV gives results for executing each example for Mozart, while Table V gives numbers for Gecode. Note that the scheduling examples have been only evaluated with Mozart.

In order to do the scheduling examples with Mozart, we needed to add a bounds propagation version of `cumulativeEF` to Mozart. We mimicked a range-equivalent version of `cumulativeEF` by using the domain propagation version on a new copy $(x'_1, \ldots, x'_n)$ of the original variables $(x_1, \ldots, x_n)$, and connecting these to the original variables through inequalities $x_i \geq x'_i$ and $x'_i \geq x_i$.

The tables contain the number of times the domain of a variable is changed (DomChg), the number of times a propagator is executed (Exec), and the runtime (wall-time) in milliseconds. The numbers for the transformed programs are given relative to the numbers of the original programs. A negative percentage means that the transformed program improves by that percentage. For example, a time-value of −50% means that the transformed program is twice as fast, whereas −90% means that the transformed programs is ten times as fast. A positive percentage is analogous.

The results show substantial improvement in execution time for the first class of benchmarks illustrating the expense of domain propagation on large equations. The number of nodes explored in each case is identical (illustrating Theorem 4.10 in action) for this and all benchmarks. Moreover the domains in this case are always identical (not just bounds-equal) as is illustrated by the number of domain changes (for both systems) and executions (for Mozart). Note that the two systems use different algorithms to implement domain-consistent linear equalities. The Gecode systems features a naive algorithm geared at small linear equations. This results in the fact that all but the two examples using small equalities (`vc-20` and `is-20`) cannot be solved in reasonable time in the non-transformed version.

The results for the second set of benchmarks show a moderate speedup, which is slightly larger for Gecode than for Mozart. This is due to the fact that the bounds version of `alldifferent` in both systems is based on a simple $O(n^2)$ algorithm presented in [Puget 1998] (with some additional improvements in Gecode). It can be expected that with a state-of-the-art implementation of a bounds propagation version of `alldifferent` (either the $O(n \log n)$ algorithm of [Puget 1998], or the linear algorithm of [Mehlhorn and Thiel 2000]), the improvement in runtime will be considerably better. Note that for Mozart the number of domain changes reduces for the larger benchmarks (`golomb-*`) indicating useless (in terms of search space) removal of internal values. Interestingly the number of executions of propagators can be greater for bounds propagation since it may require a number of executions to determine the same information as the domain version. The comparison of the two systems shows how different solvers can have markedly different internal behaviour in terms of domain changes and propagator executions.

The results for the third set of benchmarks show naturally the same behaviour for `photo-1` as the first set of examples (bounds propagation for a large linear equality), and a combination with the observations made for the second set of examples for `photo-2` (additionally, bounds propagation on `alldifferent`). It is interesting to observe that even though most constraints are concerned with expressing placement satisfaction through reification which are not transformed, the examples show already promising speedup. This suggests that even if only a fraction of the involved constraints can be optimized, our transformation is beneficial.

For the fourth class of benchmarks (only for Mozart), we obviously expect a slow down since we are mimicking a range-equivalent bounds propagation version of `cumulativeEF` using the domain version. However, bounds propagation requires less variable modifications as well as less constraint executions. This suite shows that it is worth investigating bounds propagators for `cumulativeEF` which are range-equivalent to the current domain version.

6. CONCLUSION

We have examined the propagation behaviour of domain and bounds propagators for common primitive constraints, and discovered cases where they will determine failure at the same time. By constructing theorems about how conjunctions of propagators can be built which maintain this property we are able to prove when domain and bounds propagation for a constraint system will give the same behaviour. We devised an analysis to determine where we can safely replace domain propagation by bounds propagation in a CLP(FD) program. We have illustrated a number of real programs where the analysis is able to determine weaker forms of propagators with equivalent search behaviour, and gave some evidence for the improvements possible.

The primitive constraints that we consider in this paper, while they include all of the most common constraints used in integer constraint programs, is not exhaustive. The approach can be extended to other primitive constraints by evaluating their endpoint relevance and range equivalence. Indeed the core of the paper is about establishing bounds equivalence of two sets of propagators. We do not need to restrict ourselves to bounds or domain propagators to apply the methods herein.

There is plenty more scope for understanding when one form of propagator is equivalent in strength to another. We should characterise the many global constraints available like `alldifferent` and `cumulativeEF` in terms of their propagation behaviour, and extend the analysis to handle them. The most important use of this information is probably in building more efficient versions of global constraints and recognizing where they can be used safely without increasing search space. There is also scope for finding weaker conditions that maintain bounds-equality of domains for constraints.

Other kinds of propagation are also worth considering such as value propagation or propagators for stronger notions of consistency like path consistency.

There is further scope for improving the propagators produced by the transformation. For example, consider the constraints

$$x_1 + x_2 + x_3 \leq 3, x_3 + x_4 \neq 2, \texttt{alldifferent}([x_4, x_5, x_6])$$

We can safely use the bounds propagator $bnd(x_3 + x_4 \neq 2, x_3)$ for one variable in the disequality while using the domain propagator $dom(x_3 + x_4 \neq 2, x_4)$ for the other variable.

Acknowledgements

We are grateful to Tobias Müller for support with mimicking bounds-propagation.

REFERENCES

Apt, K. 2003. *Principles of Constraint Programming.* Cambridge University Press.

Baptiste, P., Le Pape, C., and Nuijten, W. 2001. *Constraint-based Scheduling.* International Series in Operations Research & Management Science. Kluwer Academic Publishers, Boston, MA, USA.

Beasley, J. E. OR library. `http://www.ms.ic.ac.uk/info.html`.

Demoen, B., García de la Banda, M., and Stuckey, P. J. 1999. Type constraint solving for parametric and ad-hoc polymorphism. In *Proceedings of the Twenty Second Australasian Computer Science Conference*, J. Edwards, Ed. Springer-Verlag, Auckland, New Zealand, 217–228.

Dincbas, M., Simonis, H., and Van Hentenryck, P. 1990. Solving Large Combinatorial Problems in Logic Programming. *The Journal of Logic Programming 8,* 1-2 (Jan.-Mar.), 74–94.

García de la Banda, M., Hermenegildo, M., Bruynooghe, M., Dumortier, V., Janssens, G., and Simoens, W. 1996. Analysis of constraint logic programs. *ACM Transactions on Programming Languages and Systems 18,* 5, 564–614.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability.* W. H. Freeman And Company, New York, NY, USA.

Harvey, W. and Schimpf, J. 2002. Bounds consistency techniques for long linear constraints. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a workshop of CP 2002*, N. Beldiceanu, P. Brisset, M. Carlsson, F. Laburthe, M. Henz, E. Monfroy, L. Perron, and C. Schulte, Eds. Number TRA9/02. 55 Science Drive 2, Singapore 117599, 39–46.

Harvey, W. and Stuckey, P. 2003. Improving linear constraint propagation by changing constraint representation. *Constraints 7*, 173–207.

Kelly, A. D., Marriott, K., Macdonald, A., Stuckey, P. J., and Yap, R. 1998. Optimizing compilation for CLP($\mathcal{R}$). *ACM Transactions on Programming Languages and Systems 20,* 6, 1223–1250.

Lagoon, V. and Stuckey, P. J. 2001. A framework for analysis of typed logic programs. In *Proceedings of the Fifth International Symposium on Functional and Logic Programming.* Lecture Notes in Computer Science, vol. 2024. Springer-Verlag, Tokyo, Japan, 296–310.

Lesaint, D. 2002. Inferring constraint types in constraint programming. In *Proceedings of the Eigth International Conference on Principles and Practice of Constraint Programming*, P. Van Hentenryck, Ed. Lecture Notes in Computer Science, vol. 2470. Springer-Verlag, Ithaca, NY, USA, 492–507.

MARRIOTT, K. AND STUCKEY, P. J. 1998. *Programming with Constraints: an Introduction*. The MIT Press, Cambridge, MA, USA.

MARRIOTT, K. AND SØNDERGAARD, H. 1990. Analysis of constraint logic programs. In *Logic Programming: Proceedings of the 1990 North American Conference*, S. Debray and M. Hermengildo, Eds. The MIT Press, Austin, TX, USA, 531–547.

MARTIN, P. AND SHMOYS, D. B. 1996. A new approach to computing optimal schedules for the job-shop scheduling problem. In *Integer Programming and Combinatorial Optimization, 5th International IPCO Conference*, W. H. Cunningham, S. T. McCormick, and M. Queyranne, Eds. Lecture Notes in Computer Science, vol. 1084. Springer-Verlag, Vancouver, BC, Canada, 389–403.

MEHLHORN, K. AND THIEL, S. 2000. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, R. Dechter, Ed. Lecture Notes in Computer Science, vol. 1894. Springer-Verlag, Singapore, 306–319.

MOZART CONSORTIUM. 1999. The Mozart programming system. Available from `www.mozart-oz.org`.

PUGET, J.-F. 1998. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*. AAAI Press/The MIT Press, Madison, WI, USA, 359–366.

QUIMPER, C.-G., VAN BEEK, P., LÓPEZ-ORTIZ, A., GOLYNSKI, A., AND SADJAD, S. B. 2003. An efficient bounds consistency algorithm for the global cardinality constraint. In *Principles and Practice of Constraint Programming - CP 2003*, F. Rossi, Ed. Lecture Notes in Computer Science, vol. 2833. Springer-Verlag, Kinsale, Ireland, 600–614.

RÉGIN, J.-C. AND RUEHER, M. 2000. A global constraint combining a sum constraint and difference constraints. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, R. Dechter, Ed. Lecture Notes in Computer Science, vol. 1894. Springer-Verlag, Singapore, 384–395.

RÉGIN, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelth National Conference on Artificial Intelligence*. Vol. 1. AAAI Press, Seattle, WA, USA, 362–367.

VAN HENTENRYCK, P. 1989. *Constraint Satisfaction in Logic Programming*. The MIT Press, Cambridge, MA, USA.

VAN HENTENRYCK, P. 1999. *The OPL Optimization Programming Language*. The MIT Press, Cambridge, MA, USA.

VAN HENTENRYCK, P., SARASWAT, V., AND DEVILLE, Y. 1998. Design, implementation and evaluation of the constraint language cc(FD). *JLP 37,* 1–3, 139–164.

ZHANG, Y. AND YAP, R. H. C. 2000. Arc consistency on n-ary monotonic and linear constraints. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, R. Dechter, Ed. Lecture Notes in Computer Science, vol. 1894. Springer-Verlag, Singapore, 470–483.