Dynamic Analysis of Bounds versus Domain Propagation

Christian Schulte[1] and Peter J. Stuckey[2]

[1] ICT, KTH - Royal Institute of Technology, Sweden, cschulte@kth.se
[2] National ICT Australia, Victoria Laboratory, Department of Computer Science and Software Engineering University of Melbourne, Australia, pjs@cs.mu.oz.au

Abstract. Constraint propagation solvers interleave propagation (removing impossible values from variable domains) with search. Previously, Schulte and Stuckey introduced the use of static analysis to determine where in a constraint program domain propagators can be replaced by more efficient bounds propagators and still ensure that the same search space is traversed.
This paper introduces a dynamic yet considerably simpler approach to uncover the same information. The information is obtained by a linear time traversal of an analysis graph that straightforwardly reflects the properties of propagators implementing constraints. Experiments confirm that the simple dynamic method is efficient and that it can be used interleaved with search, taking advantage of the simplification of the constraint graph that arises from search.

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 dynamically analysing finite domain constraint problems 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.

Example 1. Consider the following constraints where $x_1, \ldots, x_4$ range over integer values -3 to 3 (the constraint graph is shown in Fig. 1):

$$x_1 = |x_2|, x_2 \neq x_3, 2x_3 - 3x_4 = 3, x_4 \geq x_1$$

Each constraint could be implemented using domain propagation or bounds propagation. Clearly, if each constraint uses 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?

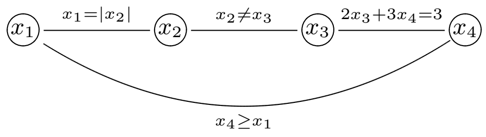

Fig. 1. Binary constraint graph for $x_1 = |x_2|, x_2 \neq x_3, 2x_3 - 3x_4 = 3, x_4 \geq x_1$

Domain propagation on $x_4 \geq x_1$ is equivalent to bounds propagation since the constraint only places upper and lower bounds on its variables. This is not the case for the remaining constraints: If $x_2 = 2$ and $x_3 \in [-3 \,..\, 3]$ then domain propagation on $x_2 \neq x_3$ determines that $x_3 \in \{-3, -2, -1, 0, 1, 3\}$ whereas bounds propagation learns nothing. Similarly if $x_1 \in \{0, 2, 3\}$ and $x_2 \in [-3 \,..\, 3]$ then domain propagation determines that $x_2 \in \{-3, -2, 0, 2, 3\}$ but bounds propagation learns nothing. From the initial set of values domain propagation determines that $x_3 \in \{-3, 0, 3\}$ and $x_4 \in \{-1, 1, 3\}$ while bounds propagation determines that $x_3 \in [-3 \,..\, 3]$ and $x_4 \in [-1 \,..\, 3]$.

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 and depends only on the variable bounds. This is in contrast to a strategy that assigns a variable to its middle domain value, or excludes its middle domain value.

Domain propagation and bounds propagation differs if changing the bounds of some variable (by search) causes change in the bounds of some variable by domain propagation which is not found by bounds propagation.

Suppose search sets $x_3 = 0$, then bounds and domain propagation of $2x_3 + 3x_4 = 1$ sets $x_4 = 1$. Bounds (and domain, as it is identical) propagation of $x_4 \geq x_1$ forces $x_1 \in [-3 \,..\, 0]$. Bounds and domain propagation on $x_1 = |x_2|$ forces $x_2 \in [-1 \,..\, 1]$. Bounds propagation on $x_2 \neq x_3$ makes no changes, resulting in a fixpoint for bounds propagation. Domain propagation on $x_2 \neq x_3$ makes $x_2 \in \{-1, 1\}$. Domain propagation on $x_1 = |x_2|$ then forces $x_1 = 1$. The resulting bounds for x_1 have changed, hence future search is affected.

But we do not need to use domain propagation for all constraints. Domain propagation on $x_2 \neq x_3$ and $x_1 = |x_2|$ is vital, as the above discussion shows. Domain propagation on $2x_2 + 3x_3 = 3$ is not required. As discussed above, the bounds of x_2 and x_3 after domain or bounds propagation are identical, and indeed we can prove this is always the case. Neither of the other constraints on x_2 and x_3 can propagate information from holes in their domains, hence the resulting propagation will be the same. $\square$

Previously [1] we introduced a static analysis of a finite domain CLP program that was able to determine when to replace domain propagators by bounds propagators without increasing search space. This paper provides a dynamic linear time analysis of the propagation graph that determines whether domain propagators can be replaced by bounds propagators without increasing search space. The approach simplifies and generalizes the previous approach.

Example 2. Consider the constraints

$$x_1 = |x_2|, x_2 \neq x_3, 2x_3 - 3x_4 = x_5, x_4 \geq x_1, x_5 \neq x_4 - 1, x_5 \geq x_2$$

where $x_1, \ldots, x_5$ range over integers from -3 to 3. Analysing this problem using the method of [1] or this paper will determine that no domain propagator can be replaced by a bounds propagator. But if search sets $x_5 = 3$ the constraints of Example 1 are obtained, since x_5 is replaced by 3 in $2x_3 - 3x_4 = x_5$, and the redundant constraints $x_5 \neq x_4 - 1$ and $x_5 \geq x_2$ are removed. A dynamic analysis can now detect that bounds propagation can be used for $2x_3 - 3x_4 = 3$ without increasing search space. □

The contributions of this paper are:

– A linear time analysis of the propagation graph that allows us to determine if we can replace bounds propagators by domain propagators without increasing search space.
– The analysis is dynamic, that is it can be run at any stage during the search. Since propagators become simpler as search proceeds this provides more scope for optimization than a static analysis before search begins.
– We show examples where our analysis detects search space equivalent replacements for both static and dynamic uses and show the possible performance benefits that arise.

2 Propagation-based Constraint Solving

This section defines our terminology for the basic components of a constraint propagation engine. In this paper we restrict ourselves to finite domain integer constraint solving. Almost all the discussion applies to other forms of finite domain constraint solving such as for sets and multisets.

Domains. A *domain* D is a complete mapping from a fixed (finite) set of variables $\mathcal{V}$ to finite sets of integers. A *false domain* D is a domain with $D(x) = \emptyset$ for some $x \in \mathcal{V}$. A variable $x \in \mathcal{V}$ is *fixed* by a domain D, if $|D(x)| = 1$. The *intersection* of domains D_1 and D_2, denoted $D_1 \sqcap D_2$, is defined by the domain $D(x) = D_1(x) \cap D_2(x)$ for all $x \in \mathcal{V}$. By $-\{x\}$ we denote the variable set $\mathcal{V} - \{x\}$.

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 $x \in \mathcal{V}$. 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 $x \in V$.

A range is a contiguous set of integers, we use *range* notation $[l\ ..\ u]$ to denote the range $\{d \in \mathbb{Z} \mid l \leq d \leq u\}$ when l and u are integers. A domain is a *range domain* if $D(x)$ is a range for all x. Let $D' = \text{range}(D)$ be the smallest range domain containing D, that is, the unique domain $D'(x) = [\inf D(x)\ ..\ \sup D(x)]$ for all $x \in \mathcal{V}$. A domain D is *bounds equivalent* to a domain D', written $D \overset{B}{=} D'$ iff $\text{range}(D) = \text{range}(D')$.

Valuations and constraints. An *integer valuation* θ is a mapping of variables to integer 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 variables appearing in a valuation. We define a valuation θ to be an element of a domain D, written $\theta \in D$, if $\theta(x_i) \in D(x_i)$ for all $x_i \in \text{vars}(\theta)$.

The *infimum* and *supremum* of an expression e with respect to a domain D are defined as $\inf_D e = \inf \ \{\theta(e) | \theta \in D\}$ and $\sup_D e = \sup \ \{\theta(e) | \theta \in D\}$.

A *constraint* c over variables $x_1, \ldots, x_n$ is a set of valuations θ such that $\text{vars}(\theta) = \{x_1, \ldots, x_n\}$. We also define $\text{vars}(c) = \{x_1, \ldots, x_n\}$.

Propagators. We will *implement* a constraint c by a set of propagators $\text{prop}(c)$ that map domains to domains. A *propagator* f is a monotonically decreasing function from domains to domains: $f(D) \sqsubseteq D$, and $f(D_1) \sqsubseteq f(D_2)$ whenever $D_1 \sqsubseteq D_2$. For the purposes of this paper we also assume that propagators are *idempotent*, that is $f(f(D)) = f(D)$ for all domains D. In fact this assumption is just required for defining the edges in the analysis graph correctly, it is not important for the actual execution.

A propagator f is *correct* for a constraint c iff $\{\theta \mid \theta \in D\} \cap c = \{\theta \mid \theta \in f(D)\} \cap c$ for all domains D. This restriction is very weak, for example the identity propagator is correct for all constraints.

The variables $\text{vars}(f)$ of a propagator f are defined as $\{v \in \mathcal{V} \mid \exists D.\ f(D)(v) \neq D(v)\} \cup \{v \in \mathcal{V} \mid \exists D_1, D_2.\ D_1 =_{-\{v\}} D_2, f(D_1) \neq_{-\{v\}} f(D_2)\}$. The set includes the variables that can change as a result of applying f, and the variables that can modify the result of f.

Example 3. For the constraint $c \equiv x_1 \leq x_2 + 1$ the function f_1 defined by $f_1(D)(x_1) = \{d \in D(x_1) \mid d \leq \sup_D x_2 + 1\}$ and $f(D)(v) = D(v), v \neq x_1$ is a correct propagator for c. Its variables are x_1 whose domain can be modified by f_1 (the first case of the definition above) and x_2 which can cause the modification of the domain of x_1 (the second case of the definition above). So $\text{vars}(f_1) = \{x_1, x_2\}$. Let $D_1(x_1) = \{1, 5, 8\}$ and $D_1(x_2) = \{1, 5\}$, then $f(D_1) = D_2$ where $D_2(x_1) = D_2(x_2) = \{1, 5\}$. The propagator is idempotent. □

A *propagation solver* $\text{solv}(F, D)$ for a set of propagators F and an initial domain D finds the greatest mutual fixpoint of all the propagators $f \in F$. In other words, $\text{solv}(F, D)$ returns a new domain defined by

$$\text{solv}(F, D) = \text{gfp}(\lambda d.\, \text{iter}(F, d))(D) \qquad \text{iter}(F, D) = \bigsqcap_{f \in F} f(D)$$

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

Domain and bounds propagators. A consistency notion C gives a condition on domains with respect to constraints. A set of propagators F maintains C-*consistency* for a constraint c, if for domain D where $f(D) = D, f \in F$ is always C consistent for c. Many propagators in practice are designed to maintain some form of consistency: usually domain or bounds.

The most successful consistency technique is *arc consistency* [2], which ensures that for each binary constraint, every value in the domain of the first variable has a supporting value in the domain of the second variable that satisfies the constraint. Arc consistency can be naturally extended to constraints of more than two variables to give *domain consistency.* A domain D is *domain consistent* for a constraint c if D is the least domain containing all solutions $\theta \in D$ of c, that is, there does not exist $D' \sqsubset D$ such that $\theta \in D \wedge \theta \in c \rightarrow \theta \in D'$.

Define the *domain propagator* $\mathrm{dom}(c)$, for a constraint c as

$$\begin{array}{ll} \mathrm{dom}(c)(D)(x) = \{\theta(x) \mid \theta \in D \wedge \theta \in c\} & \text{where } x \in \mathrm{vars}(c) \\ \mathrm{dom}(c)(D)(x) = D(x) & \text{otherwise} \end{array}$$

Bounds consistency relaxes the consistency requirement to apply only to the lower and upper bounds of each variable x. There are a number of different notions of bounds consistency [3], we give the two most common here.

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

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

A propagator f is a *bounds propagator* if it only relies on bounds and creates new bounds

$$\forall D.\ f(D) = \mathrm{range}(f(\mathrm{range}(D))) \sqcap D$$

We can define bounds propagators for the two consistency notions above. A bounds($\mathbb{Z}$) *propagator*, $\mathrm{zbnd}(c)$ for a constraint c ensures that $\mathrm{zbnd}(c)(D)$ is bounds($\mathbb{Z}$) consistent with c, while a bounds($\mathbb{R}$) *propagator*, $\mathrm{rbnd}(c)$ ensures bounds($\mathbb{R}$) consistency.

3 An Abstraction of Propagation

The aim of this paper is to find where we can replace a propagator f by a bounds propagator f^B without changing the search space, under the assumption that

$$\forall D.\ f(\mathrm{range}(D))) \overset{B}{=} f^B(D)$$

That is, applied to range domains the propagators give the same bounds. Note that if $f = \mathrm{dom}(c)$ and $f^B = \mathrm{zbnd}(c)$ then this property holds. We will not attempt to replace domain propagators by bounds($\mathbb{R}$) propagators since the property does not hold.

Example 4. Consider the constraint $c \equiv x = 3y + 5z$, and the range domain $D(x) = [2 \mathrel{..} 7]$, $D(y) = [0 \mathrel{..} 2]$ and $D(z) = [-1 \mathrel{..} 2]$, then $\mathrm{dom}(c)(D)(x) = \{3, 5, 6\}$ while $\mathrm{rbnd}(c)(D)(x) = [2 \mathrel{..} 7]$. The bounds are different. $\square$

In order to detect that we can replace domain propagators by bounds propagators, we build an analysis graph that shows how each propagator reacts to holes and creates holes in the domain of its variables. The analysis graph is in some sense an abstraction of the constraint (hyper)graph where an edge is an abstract propagator. Analysis of the graph corresponds to executing the abstract propagators to fixpoint, hence is an abstract propagation process.

The nodes of an analysis graph G are labelled by variables $v \in \mathcal{V}$, as well as the special nodes *source* $\oplus$ and *sink* $\ominus$. The analysis graph G for a set of propagators F contains directed edges for each propagator f as follows:

- An edge $x \xrightarrow{f} y$ between two variables x and y labelled by a propagator f indicates that f can propagate holes in the domain of x to the domain of y.
  There is an edge $x \xrightarrow{f} y$ in G, iff there exist domains D, D' and variables $S \subseteq \mathcal{V}$ with $D' =_{\mathcal{V}-S} D$, $D \overset{B}{=} D'$, $D'(x) \neq D(x)$, and $x \in S$ such that $f(D')(y) \neq f(D)(y) \cap D'(y)$. That is, D' differs from D only because of the removal of internal values for variables S including x.
- An edge $x \xrightarrow{f} \ominus$ between variable x and the sink indicates that by propagating f, holes in the domain of x can cause bounds changes on other variables.
  There is an edge $x \xrightarrow{f} \ominus$ in G, iff there exist domains D, D' and variables $S \subseteq V$ with $D' =_{V-S} D$, $D \overset{B}{=} D'$, $D'(x) \neq D(x)$, and $x \in S$ such that $f(D') \overset{B}{\neq} f(D)$.
- An edge $\oplus \xrightarrow{f} x$ between the source and variable x indicates that the propagator f can create holes in the domain of x from a range domain.
  There is an edge $\oplus \xrightarrow{f} x$ in G, iff there exists a range domain D (that is, $D = \text{range}(D)$) such that $f(D)(x) \neq_{\{x\}} \text{range}(f(D))(x)$. That is, applying f to a range domain D can create a hole in the domain of x.

As an example, let us consider the edges in the analysis graph for some common domain propagators (a full table is given in Appendix A):

dom$(x = y + k)$ $(x, y \in \mathcal{V},\ k \in \mathbb{Z})$: $\{x \xrightarrow{f} y, y \xrightarrow{f} x\}$. Holes are propagated, but neither created nor converted to bounds.

dom$(x \neq y)$ $(x, y \in \mathcal{V})$: $\{\oplus \xrightarrow{f} x, \oplus \xrightarrow{f} y\}$. Holes are not propagated, but created.

dom$(x = k \times y)$ $(x, y \in \mathcal{V}, k \in \mathbb{Z})$: $\{x \xrightarrow{f} y, y \xrightarrow{f} x, \oplus \xrightarrow{f} x\}$. Holes are propagated and holes for x are created.

dom$(x = |y|)$ $(x, y \in \mathcal{V})$: $\{x \xrightarrow{f} y, y \xrightarrow{f} x, \oplus \xrightarrow{f} y, y \xrightarrow{f} \ominus\}$. Holes are transmitted, holes for y are created (by bounds of x), and holes in y can change bounds (for x).

dom$(\sum_{i=1}^{n} x_i \leq k)$ $(x_i \in \mathcal{V},\ k \in \mathbb{Z})$: $\{\}$. No holes are created or transmitted.

dom$(\sum_{i=1}^{n} x_i = k)$ $(x_i \in \mathcal{V},\ k \in \mathbb{Z})$: $\{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq n\}$. Holes are transmitted between each pair.

dom$(b \Leftrightarrow x = y)$ $(b, x, y \in \mathcal{V},\ b$ Boolean$)$: $\{x \xrightarrow{f} y, y \xrightarrow{f} x, \oplus \xrightarrow{f} x, \oplus \xrightarrow{f} y, x \xrightarrow{f} \ominus, y \xrightarrow{f} \ominus\}$. Unsurprisingly the union of $x = y$ and $x \neq y$, except that holes

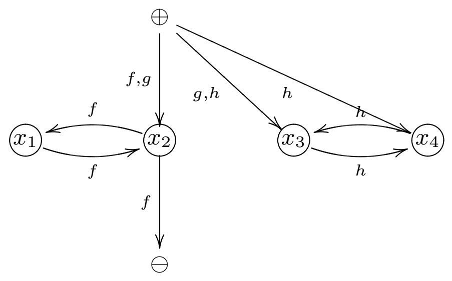

Fig. 2. Analysis graph for $x_1 = |x_2|$, $x_2 \neq x_3$, $2x_3 + 3x_4 = 3$, $x_4 \geq x_1$.

in x and y can create bounds changes in b. For example, $D(x) = \{1, 3, 5\}$ and $D(y) = \{2, 4, 6\}$ yields $b = 0$.

dom(`alldifferent`$(x_1, \ldots, x_n)$) $(x_i \in \mathcal{V})$: $\{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq n\} \cup \{\oplus \xrightarrow{f} x_i, x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq n\}$. The propagator can do everything. But we should be careful, we do have a bounds($\mathbb{Z}$) propagator that will give the same bounds, if no other propagator causes holes in the domains.

dom$(x = \min(y, z))$ $(x, y, z \in \mathbb{Z})$: $\{x \xrightarrow{f} y, y \xrightarrow{f} x, x \xrightarrow{f} z, z \xrightarrow{f} x, y \xrightarrow{f} \ominus, z \xrightarrow{f} \ominus\}$. There is no direct transmission from y to z, and no changes of ranges from holes. Notice that for $D(x) = \{1, 3, 4\}$, $D(y) = \{1, 3, 5\}$, $D(z) = \{2, 4, 6\}$ the constraint includes the solutions $(1, 1, 2), (3, 3, 4), (4, 5, 4), (3, 3, 6)$. But changing $D(z)$ to $\{2, 6\}$ changes the upper bound of x.

Example 5. The analysis graph for the problem of Example 1 is shown in Figure 2, where $f = \text{dom}(x_1 = |x_2|)$, $g = \text{dom}(x_2 \neq x_3)$, $h = \text{dom}(2x_3 + 3x_4 = 3$ and $\text{dom}(x_4 \geq x_1)$ does not generate any edges. The reasoning in Example 1 is now explicitly viewable. The path from $\oplus \xrightarrow{g} x_2 \xrightarrow{f} \ominus$ shows that holes created by g can cause bounds to change through f as illustrated in Example 1.

While h can create new holes, the arcs from $\oplus$ to x_3 and x_4, these holes can never change the bounds of a variable (reach $\ominus$). Hence h can be replaced by a bounds($\mathbb{Z}$) propagator, without changing propagation. □

4 Main Result

A path from $\oplus$ to $\ominus$ is evidence of where bounds information can create holes in domains, and where holes then can change bounds. We must keep track of the holes in the domains in order to have accurate bounds information.

Theorem 1. *Let G be the analysis graph for a set of propagators F. Let $B \subseteq F$ be the set of propagators such that G contains no paths from $\oplus$ to $\ominus$ labelled with two or more propagators including f. Then $F' = \{f^B \mid f \in B\} \cup \{f \mid f \in F - B\}$ is such that* $\text{solv}(F, D) \overset{B}{=} \text{solv}(F', D)$ *for all range domains D.*

Proof. The proof is by induction. Let f_1, f_2, $\ldots$, f_n be the sequence of propagators applied in calculating $\mathrm{solv}(F, D)$. Let

$$D = D_0, f_1(D_0) = D_1, f_2(D_1) = D_2, \ldots, f_n(D_{n-1}) = D_n = \mathrm{solv}(F, D)$$

We let g_i be the corresponding propagator to f_i in F', that is $g_i = f_i$ if $f_i \in F - B$ and $g_i = f_i^B$ if $f_i \in B$. Define

$$D = D'_0, g_1(D'_0) = D'_1, g_2(D'_1) = D'_2, \ldots, g_n(D'_{n-1}) = D'_n$$

be the analogous sequence of propagators in F'. Note that since $\forall D.\ f_i(D) \sqsubseteq g_i(D)$ we have that $D_i \sqsubseteq D'_i$. We show that $D_i \overset{B}{=} D'_i$ for $0 \leq i \leq n$.

We show by induction: for each $0 \leq i \leq n$ that $D_i \overset{B}{=} D'_i$, and for each $v \in \mathcal{V}$ where $D_i(v)$ is not a range then there is a path from $\oplus$ to v in the analysis graph.

Clearly the induction hypothesis holds for $D_0 = D'_0$. Assume the hypothesis holds for $0 \leq i < K$.

Suppose to the contrary that $D_K \overset{B}{\neq} D'_K$. If $D_{K-1} =_{\mathrm{vars}(f_K)} \mathrm{range}(D_{K-1})$ then $D_K = f_K(D_{K-1}) = f_K(\mathrm{range}(D_{K-1})) \overset{B}{=} g_K(D_{K-1}) \overset{B}{=} g_k(D'_{K-1}) = D'_K$. Contradiction. Hence there exists $v \in \mathrm{vars}(f_K)$ such that $D_{K-1}(v)$ is not a range. By the induction hypothesis, there is a path from $\oplus$ to v. And by definition $v \overset{f_K}{\to} \ominus$ is in the analysis graph G. The witnesses are $D' = D_{K-1}$, $D = D'_{K-1}$, $S = \mathrm{vars}(f_K)$ and $x = v$. Hence all propagators modifying the interior of the domain of v are either the same propagator f_K or are not in B. In the first case, since f_K is idempotent $f_k(D_{K-1}) = f_K(\mathrm{range}(D_{K-1})$ as no other propagators have changed the interior of the domains of $\mathrm{vars}(f_K)$. Thus $D_K = f_K(D_{K-1}) = f_K(\mathrm{range}(D_{K-1})) \overset{B}{=} g_K(D'_{K-1}) = D'_K$. Contradiction. In the second case since all propagators acting on the interior of domains of $\mathrm{vars}(f_K)$ are in $F - B$ we have that $D_{K-1} =_{\mathrm{vars}(f_K)} D'_{K-1}$, and $f_K = g_K$ hence $D_K \overset{B}{=} D'_K$. Contradiction. As a result we have that $D_K \overset{B}{=} D'_K$.

Suppose that $D_K(v)$ is not a range, and $D_K(v) \neq D_{K-1}(v)$

If $D_{K-1} =_{\mathrm{vars}(f_K)} \mathrm{range}(D_{K-1})$ then we have an edge $\oplus \overset{f_K}{\to} v$ is the analysis graph G. The witness is $D = \mathrm{range}(D_{K-1})$ and $x = v$, since $D = \mathrm{range}(D)$ and $f_K(D)(v) = f_K(D_{K-1})(v) \neq \mathrm{range}(f_K(D))(v)$.

Otherwise $D_{K-1} \neq_{\mathrm{vars}(f_K)} \mathrm{range}(D_{K-1})$, and so there exists $u \in \mathrm{vars}(f_K)$ where $D_{K-1}(u)$ is not a range. Then we have an edge $u \overset{f_K}{\to} v$ in the analysis graph G. The witnesses are $D' = D_{K-1}$ and $D = \mathrm{range}(D_{K-1})$, $S = \mathrm{vars}(f_K)$, $x = u$ and $y = v$. By the induction hypothesis there is a path from $\oplus$ to u in the analysis graph, and hence also to v.

As a result of the proof by induction we have that $D_n \overset{B}{=} D'_n$. Then since D_n is a fixpoint for all f_i, and since g_i where they differ from f_i only depend on bounds, we have that D_n is a fixpoint for all g_i. Now $\mathrm{solv}(F', D)$ is the greatest fixpoint of F' less than D and D_n is such a fixpoint we have that $D_n \sqsubseteq \mathrm{solv}(F', D) \sqsubseteq D'_n$ and hence $\mathrm{solv}(F, D) = D_n \overset{B}{=} \mathrm{solv}(F', D)$.

□

Note that the proof can be applied for *non-range* domains D, by adding artificial propagators f that remove the internal values of range(D) to give D. In effect we add edges $\oplus \xrightarrow{f} v$ for each v where $D(v) \neq \text{range}(D)(v)$.

Importantly the theorem is based on propagators rather than constraints, hence we might have bounds propagators in the original propagators F we are trying to improve.

Example 6. Consider propagators for the SEND+MORE=MONEY problem: $f = \text{dom}(\texttt{alldifferent}(S, E, N, D, M, O, R, Y))$, a large linear bounds propagator rbnd$(SEND + MORE = MONEY)$, dom$(S > 0)$, and dom$(M > 0)$. The only edges are $\{x \xrightarrow{f} y \mid x, y \in \{S, E, N, D, M, O, R, Y\}, x \neq y\} \cup \{\oplus \xrightarrow{f} x, x \xrightarrow{f} \ominus \mid x \in \{S, E, N, D, M, O, R, Y\}\}$. All propagators can be replaced by bounds propagators. If the long linear constraint used domain propagation the propagator for the `alldifferent` constraint could not be improved. □

In order to replace propagators by equivalent propagators we have to take into account the constraints that will be added by search. Edges are added corresponding to the behaviour of the search procedure. If search relies:

- on bounds information to make decisions and only adds bounds constraints, no edges are added (e.g. standard `labelling`$(x_1, \ldots, x_n)$);
- on all domain information to make decisions but only add bounds constraints, $\{x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq n\}$ are added (e.g. `labellingff`$(x_1, \ldots, x_n)$ for first-fail labelling);
- on all domain information and may add constraints that add holes to domains, $\{\oplus \xrightarrow{f} x_i, x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq n\}$ are added (e.g. middle out labelling `labellingmid`$(x_1, \ldots, x_n)$).

5 Finding which Propagators to Replace

In order to use Theorem 1 we need to determine which propagators appear on paths from $\oplus$ to $\ominus$, involving at least two propagators. Rather than track (a possibly exponential number of) paths explicitly, we mark each variable x by the propagators on paths from $\oplus$ to x, and by the propagators on paths from x to $\ominus$. We can check each edge for a propagator f to see whether it causes f to be on a path from $\oplus$ to $\ominus$, involving at least two propagators.

The algorithm is shown in Figure 3. Assuming that munion(m_1,m_2) is simply defined as $m_1 \cup m_2$, the propagators on a path from $\oplus$ to n are stored in source$[n]$, while sink$[n]$ holds the propagators on a path from n to $\ominus$. The forward marking starts from all variables adjacent to $\oplus$ and marks them, and then follows any edges to continue marking. It checks if the variable has been marked previously with the current set and if so immediately returns. The backward marking works analogously. Finally the new propagator set F' is constructed by checking each edge for propagator f, and if it takes part in a path from $\oplus$ to $\ominus$ involving at least two propagators, adding the original version f to F' otherwise adding the bounds version f^B.

```
    munion($m_1$,$m_2$)
        **if** ($|m_1 \cup m_2| > 1$) **return** $F$ **else return** $m_1 \cup m_2$

    forward($x$,$m$)
        **if** ($m \subseteq \text{source}[x]$) **return**
        $\text{source}[x] \leftarrow$ munion($\text{source}[x]$,$m$)
        **for** ($x \xrightarrow{g} y \in G$)
            forward($y$,munion($\text{source}[x]$,$\{g\}$))

    backward($x$,$m$)
        **if** ($m \subseteq \text{sink}[x]$) **return**
        $\text{sink}[x] \leftarrow$ munion($\text{sink}[x]$,$m$)
        **for** ($y \xrightarrow{g} x \in G$)
            backward($y$,munion($\text{sink}[x]$,$\{g\}$))

    domain($f$,$G$)
        **return** $\exists n_1 \xrightarrow{f} n_2 \in G.\ |\text{munion}(\text{source}[n_1], \text{munion}(\{f\}, \text{sink}[n_2]))| > 1$

analyse($F$)
    **let** $G$ be the analysis graph for $F$
    **for** ($n \in \mathcal{V} \cup \{\oplus, \ominus\}$)
        $\text{source}[n] \leftarrow \text{sink}[n] \leftarrow \emptyset$
    **for** ($\oplus \xrightarrow{f} x \in G$)
        forward($x$,$\{f\}$)
    **for** ($x \xrightarrow{f} \ominus \in G$)
        backward($x$,$\{f\}$)
    **return** $\{f \mid f \in F \wedge \text{domain}(f, G)\} \cup \{f^B \mid f \in F \wedge \neg\text{domain}(f, G)\}$
```

Fig. 3. Propagation analysis of the set of propagators F

Theorem 2. *Let G be the analysis graph for F. Let B be the set of propagators $f \in F$ such that G contains no paths from $\oplus$ to $\ominus$ labelled with two or more propagators including f. Then analyse(F) $= \{f^B \mid f \in B\} \cup \{f \mid f \in F - B\}$ and the complexity of analyse(F) is $O(G)$.*

Proof. (Sketch) Under the assumption that munion(m_1,m_2) is simply defined as $m_1 \cup m_2$ it is easy to see that variables source[n] and sink[n] contain the set of propagators appearing in paths from $\oplus$ to n and n to $\ominus$ respectively. The final test $|\text{source}[n_1] \cup \{f\} \cup \text{sink}[n_1]| \geq 2$ correctly determines if f appears on a path from $\oplus$ to $\ominus$ involving at least two propagators.

Now consider the actual definition of munion(m_1,m_2). This is in effect an abstraction of the original algorithm where all sets of cardinality greater than 1 are replaced by F. This does not change the result of the final test. For the test to fail, source[n_1] and sink[n_1] are either $\{f\}$ or $\emptyset$, and these results are maintained by the actual definition of munion(m_1,m_2). For the test to pass $|\text{source}[n_1] \cup \{f\} \cup \text{sink}[n_1]| > 1$ and hence also $|\text{munion}(\text{source}[n_1], \text{munion}(\{f\}, \text{sink}[n_2]))| > 1$. Hence the algorithm is correct.

The complexity result follows since forward can only update source$[n]$ at most twice, after which source$[n] = F$ and all further calls immediately return. Hence the complexity of all calls to forward is $O(G)$. The same reasoning applies to backward, and hence to analyse. □

The astute reader will have noticed that, while analyse is linear in the size of the analysis graph, the analysis graph may be quadratically larger in size than the constraint graph, since some propagators add edges $\{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq n\}$. This is fixed by replacing the edges $\{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq n\}$ by the edges $\{x_i \xrightarrow{f} z, z \xrightarrow{f} x_i \mid 1 \leq i \leq n\}$ where z is a new variable. The resulting analysis graph is linear in the size of the constraint graph, and gives the same results as the original graph.

Implementation. The algorithm in Figure 3 has been implemented in Gecode, but the decisions made in the implementation should readily carry over to other constraint programming systems.

While treatment of variables is generic in the analysis algorithm, the way how propagators are analysed depends on the particular propagator. Propagators are implemented as objects in Gecode. Propagators provide methods for propagation, creation, deletion, and so on. For analysis, we add a `analyse` method that can be implemented for each individual propagator: execution of the method adds the edges for the propagator to the analysis graph.

The values of source$[x]$ and sink$[x]$ are directly stored in the variable x. Rather than storing a set of propagators F for source$[x]$ and sink$[x]$, it is sufficient to use a pointer to a propagator f (if $F = \{f\}$) and two special marks $\langle 0 \rangle$ ($|F| = 0$) and $\langle 2 \rangle$ ($|F| \geq 2$). Then munion(m_1, m_2) returns m as follows: if $m_1 = m_2$ then $m = m_1$; if $m_1 = \langle 2 \rangle$ or $m_2 = \langle 2 \rangle$ then $m = \langle 2 \rangle$; if $m_1 = \langle 0 \rangle$ then $m = m_2$; if $m_2 = \langle 0 \rangle$ then $m = m_1$.

6 Experimental Evaluation

All experiments use Gecode, a C++-based constraint programming library [5]. Gecode is one of the fastest constraint programming systems currently available, benchmarks comparing Gecode to other systems are available from Gecode's webpage. The version used here corresponds to Gecode 2.1.1. Gecode has been compiled with the Microsoft Visual Studio Express Edition 2008 (32 bit).

All examples have been run on a Mac Pro with two 2.8 GHz Quad Core Xeon 5400 CPUs and 8192 MB main memory running 64 bit Windows Vista. Runtimes are the average of 25 runs with a coefficient of deviation less than 3% for all benchmarks.

Static analysis. Table 1 shows the runtime (time, in milliseconds), which percentage of the runtime is spent on the analysis in the optimized case, and the number of nodes during search (as to be expected, the same for both). Examples with a – as entry have been stopped after a runtime of one hour.

Table 1. Static analysis

Example	original	optimized		both
	time	runtime	analysis	nodes
`is-20`	127.80	0.05 (−100.0%)	15.1%	13
`is-40`	–	0.16 (−100.0%)	17.5%	28
`vc-20`	87.68	0.03 (−100.0%)	27.8%	4
`vc-40`	–	0.08 (−100.0%)	36.9%	6
`photo-eq`	890.40	429.24 (−51.8%)	0.0%	5 472
`photo-lq`	714.48	78.80 (−89.0%)	0.0%	10 350
`money`	0.02	0.02 (−18.0%)	5.6%	4
`donald`	21.72	21.22 (−2.3%)	0.0%	5 788
`magic-5`	1 324.88	1 103.92 (−16.7%)	0.0%	89 016

The examples `is-`n (independent sets) and `vc-`n (vertex cover) for random graphs with n nodes are modeled in the natural way using Boolean variables. The constraints are all inequalities except the objective function which is defined using a large linear equation with unit coefficients (optimized by analysis). `photo-*` is a simple placement problem and use reified 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-eq` uses reified linear equations, `photo-lq` uses reified linear inequalities to express preferences. Analysis shows for `photo-eq` that bounds propagation can be used on the large linear equation. For `photo-lq`, bounds propagation can also used for the single occurring `alldifferent` constraint. The well-known examples `money` (see Example 6), `donald` ($DONALD + GERALD = ROBERT$), and magic square `magic-5` use bounds propagation for linear equations with more than three variables. Analysis shows that bounds propagation can be used for the single `alldifferent` constraint in each example.

The analysis is run before evaluating solv for the first time (such that infeasible domain propagators could be optimized away). The benefit of the analysis clearly outweighs its cost (for already medium sized examples the cost is zero). This is true for the expensive (exponential) and often infeasible domain propagators for long linear equations (for example, `is-`n and `vc-`n) but also for feasible domain propagators such as `alldifferent`.

Dynamic analysis and analysis cost. In the following we evaluate a variation of the analysis in order to assess its cost and benefit. We assume that the feasibility of domain propagation is classified as follows. For a constraint c (or for a propagator f implementing c) a predicate feasible(c) holds, iff it is feasible (sufficiently efficient) to use a domain propagator for c. For example, one could define feasible($\sum_{i=1}^{n} a_i x_i = d$) to hold iff $n \leq 3$, and feasible(`alldifferent`($x_1, \ldots, x_n$)) to always hold.

Initially, all constraints are propagated by bounds propagators. During search, propagators might become feasible (e.g., some x_i in $\sum_{i=1}^{n} a_i x_i = d$ become fixed).

Table 2. Dynamic analysis and analysis cost

Example	$n = 1$		$n = 5$		$n = 10$		$n = 25$	
	nodes	time	nodes	time	nodes	time	nodes	time
(a) analysis with optimization								
`alpha`	−80.7%	−40.7%	−72.8%	−43.0%	−68.4%	−9.9%	−32.6%	−4.5%
`money-c`	±0.0%	+45.6%	±0.0%	+11.0%	±0.0%	+5.9%	±0.0%	+3.5%
`donald-c`	−7.5%	+117.7%	−6.5%	+37.9%	−2.8%	+25.0%	−0.9%	+12.6%
`magic-4`	−15.2%	+485.3%	−6.5%	+314.0%	−3.4%	+209.1%	−1.7%	+120.5%
(b) only analysis								
`alpha`	±0.0%	+91.8%	±0.0%	+20.8%	±0.0%	+10.7%	±0.0%	+5.1%
`money-c`	±0.0%	+43.0%	±0.0%	+10.6%	±0.0%	+6.0%	±0.0%	+2.5%
`donald-c`	±0.0%	+99.8%	±0.0%	+22.7%	±0.0%	+12.7%	±0.0%	+4.7%
`magic-4`	±0.0%	+94.7%	±0.0%	+20.8%	±0.0%	+10.9%	±0.0%	+5.3%

We are going to use the analysis to replace a bounds propagator by a domain propagator, if it is feasible and the analysis shows that domain propagation might be beneficial. Hence, we construct the analysis graph G as follows: if the propagator is a domain propagator, the edges are entered as before. If the propagator is a feasible bounds propagator for the constraint c, the edges for $\mathrm{dom}(c)$ are entered. After running the analysis phase, domain propagators are replaced by bounds propagators if possible as before. If $\mathsf{domain}(f, G)$ holds for a feasible bounds propagator f, it is replaced by a corresponding domain propagator.

By this, only bounds propagators that are feasible *and can potentially improve propagation* are replaced by domain propagators. Just using feasibility alone would, in all benchmark examples discussed above, immediately replace the bounds propagator for `alldifferent` by a domain propagator even though this is useless. Note that as search proceeds, a bounds propagator for a constraint can be replaced by a domain propagator when becoming feasible, and later be replaced by a bounds propagator when the analysis finds that bounds propagation is sufficient.

Table 2 shows the runtime and the number of nodes during search relative to execution of the examples without running any analysis and using bounds propagators. The analysis is run every n-th time before solv is computed by the solver, where for (a) bounds and domain propagators are replaced, while for (b) the analysis results are ignored (measuring analysis cost). It is important that the analysis is run before solv is evaluated as the replacement of bounds by domain propagators might require re-evaluation of solv. The examples all use a `alldifferent` constraint and some linear equations (`money-c` and `donald-c` use several linear equations for a model using carries in the letter equation).

Clearly, running the analysis before every evaluation of solv is infeasible, however running it every 10 times reduces the overhead to around 10%: that means the analysis is efficient enough to be actually run dynamically. It may be that an incremental version of the analysis could reduce this overhead substantially. In cases where replacing bounds by domain propagators is useful as the search space shrinks, the additional cost of domain propagation might still be too high.

There is at least some evidence (`alpha`) that dynamic analysis can be beneficial, and we have just scratched the surface of possibilities for automatic selection of propagation style.

7 Conclusion and Related Work

The original work on analysing when domain propagation could be replaced by bounds propagation [1] worked in a completely different way. Propagators were classified as *bounds-preserving*: meaning that on range domains they always gave range domains; and *endpoint-relevant*: meaning that the bounds resulting from applying the propagator only depended on the bounds it was applied to. Bounds preserving propagator are propagators with no edges $\oplus \xrightarrow{f} x$, while endpoint-relevant propagators are propagators with no edges $x \xrightarrow{f} \ominus$. Two analyses were undertaken to find (Boolean) bounds preservation and endpoint relevant descriptions for the context of each constraint. Each constraint was then given the appropriate propagator by examining its context. The algorithms used in the approach are $O(nm)$ where n is the size of the constraint graph and m is the number of constraints. The analysis is substantially more complicated to implement than the approach in this paper, and indeed was never implemented.

The approach of this paper is

- considerably simpler, easier to prove, and implemented;
- $O(n)$ where n is the size of the constraint graph; and
- more expressive, although this does not lead to more replacement of domain propagators by bounds propagators. An example is the description for $\text{dom}(x_1 = |x_2|)$ which in the new approach tracks the behaviour of x_1 and x_2 more accurately than is possible in the old approach.

As future work we will consider proving and implementing a stronger version of Theorem 1 where we let B be the set of all propagators where there is no path from $\oplus$ to $\ominus$ where adjacent edges have to be from different propagators.

Example 7. Consider the propagators $f = \text{dom}(x_1 = x_2)$, $g = \text{dom}(x_2 = |x_3|)$, and $h = \text{dom}(x_3 = x_4)$, which generate the analysis graph $x_1 \xrightarrow{f} x_2$, $x_2 \xrightarrow{f} x_1$, $\oplus \xrightarrow{g} x_2$, $x_2 \xrightarrow{g} x_3$, $x_3 \xrightarrow{g} x_2$, $x_3 \xrightarrow{g} \ominus$, $x_3 \xrightarrow{h} x_4$, $x_4 \xrightarrow{h} x_3$. The analysis detects that nothing can be a bounds propagator. But indeed all could be replaced because any holes generated by g are only fed back to itself, and hence cannot change bounds. There are no alternating paths from $\oplus$ to $\ominus$. □

References

1. Schulte, C., Stuckey, P.J.: When do bounds and domain propagation lead to the same search space? ACM Trans. Program. Lang. Syst. **27**(3) (May 2005) 388–425
2. Mackworth, A.K.: Consistency in networks of relations. Artificial Intelligence **8**(1) (1977) 99–118

3. Choi, C.W., Harvey, W., Lee, J.H.M., Stuckey, P.J.: Finite domain bounds consistency revisited. In: AI 2006: Advances in Artificial Intelligence. Volume 4304 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany (2006) 49–58
4. Puget, J.F., Leconte, M.: Beyond the glass box: Constraints as objects. In Lloyd, J., ed.: Proceedings of the International Symposium on Logic Programming, Portland, OR, USA, The MIT Press (December 1995) 513–527
5. Gecode Team: Gecode: Generic constraint development environment (2006) Available from `http://www.gecode.org`.

A The Analysis Graph

Analysis graph edges for primitive constraints and some labellings. The last column shows whether a bounds($\mathbb{Z}$) propagator for the constraint is commonly available.

Constraint	G	zbnd
$\sum_{i=1}^n a_i x_i \leq d$	$\emptyset$	✔
$x_0 = d$	$\emptyset$	✔
$a_1 x_1 + a_2 x_2 = d$, $\lvert a_i \rvert = 1$	$\{x_1 \xrightarrow{f} x_2,\ x_2 \xrightarrow{f} x_1\}$	✔
$a_1 x_1 + a_2 x_2 = d$	$\{x_1 \xrightarrow{f} x_2,\ x_2 \xrightarrow{f} x_1,\ \oplus \xrightarrow{f} x_1,\ \oplus \xrightarrow{f} x_2\}$	✔
$\sum_{i=1}^n a_i x_i = d, n > 2,$	$\{\oplus \xrightarrow{f}, x_i, x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq n\} \cup \{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq n\}$	✘
$\sum_{i=1}^n a_i x_i = d$, $n > 2$, $\lvert a_i \rvert = 1$	$\{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq n\}$	✔
$\sum_{i=1}^n a_i x_i \neq d$	$\{\oplus \xrightarrow{f} x_i \mid 1 \leq i \leq n\}$	✔
$x_0 \Leftrightarrow \sum_{i=1}^n a_i x_i \leq d$	$\emptyset$	✔
$x_0 \Leftrightarrow \sum_{i=1}^n a_i x_i = d$	$\{\oplus \xrightarrow{f} x_i, x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq n\} \cup \{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq n\}$	✘
$x_1 = \neg x_2$	$\emptyset$	✔
$x_1 = (x_2 \ \&\& \ x_3)$	$\emptyset$	✔
$x_1 = (x_2 \Vert x_3)$	$\emptyset$	✔
$x_1 = (x_2 \Rightarrow x_3)$	$\emptyset$	✔
$x_1 = (x_2 \Leftrightarrow x_3)$	$\emptyset$	✔
$x_1 = x_2 \times x_3$	$\{\oplus \xrightarrow{f} x_i, x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq 3\} \cup \{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq 3\}$	✘
$x_1 = x_2 \times x_2 \wedge x_2 \geq 0$	$\{x_1 \xrightarrow{f} x_2,\ x_2 \xrightarrow{f} x_1,\ \oplus \xrightarrow{f} x_1\ \}$	✔
$x_1 = x_2 \times x_2$	$\{x_1 \xrightarrow{f} x_2, x_2 \xrightarrow{f} x_1, \oplus \xrightarrow{f} x_1, x_2 \xrightarrow{f} \ominus\}$	✔
$x_1 = \lvert x_2 \rvert$	$\{x_1 \xrightarrow{f} x_2, x_2 \xrightarrow{f} x_1, \oplus \xrightarrow{f} x_2, x_2 \xrightarrow{f} \ominus\}$	✔
$x_0 = \texttt{min}(\{x_1, \ldots, x_n\})$	$\{x_0 \xrightarrow{f} x_j, x_j \xrightarrow{f} x_0 \mid 1 \leq j \leq n\} \cup \{x_i \xrightarrow{f} \ominus \mid 0 \leq i \leq n\}$	✔
$\texttt{alldifferent}(x_1, \ldots, x_n)$	$\{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq n\} \cup \{\oplus \xrightarrow{f} x_i, x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq n\}$	✔
$\texttt{default}(x_1, \ldots, x_n)$	$\{\oplus \xrightarrow{f} x_i, x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq n\} \cup \{x_i \xrightarrow{f} x_j \mid 1 \leq i \neq j \leq n\}$	✘
$\texttt{labelling}(x_1, \ldots, x_n)$	$\emptyset$	
$\texttt{labellingff}(x_1, \ldots, x_n)$	$\{x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq n\}$	
$\texttt{labellingmid}(x_1, \ldots, x_n)$	$\{\oplus \xrightarrow{f} x_i, x_i \xrightarrow{f} \ominus \mid 1 \leq i \leq n\}$	