Learning general constraints in CSP (long version)

Michael Veksler Ofer Strichman

Information Systems Engineering, Technion, Haifa, Israel
mveksler@tx.technion.ac.il ofers@ie.technion.ac.il

Abstract. We present a new learning scheme for CSP solvers, which is based on learning (general) constraints rather than generalized no-goods or signed-clauses that were used in the past. The new scheme is integrated in a conflict-analysis algorithm reminiscent of a modern systematic SAT solver: it traverses backwards the conflict graph and gradually builds an asserting conflict constraint. This construction is based on new inference rules that are tailored for various pairs of constraints types, e.g., $x \leq y_1 + k_1$ and $x \geq y_2 + k_2$, or $y_1 \leq x$ and $[x, y_2] \not\subseteq [a, b]$. The learned constraint is stronger than what can be learned via signed resolution. Our experiments show clear advantage over the state-of-the-art solver MISTRAL in most types of constraints, averaging 25% reduction in fails (time-out or memory-out), 29% reduction in run-time of instances that both engines solved, and 95.9% average reduction in the number of backtracks, when measured on the last CSP competition (2009) benchmarks that include inequality constraints (a total of 2162 benchmarks).

1 Introduction

The ability of CSP solvers to learn new constraints during the solving process possibly shortens run-time by an exponential factor (see, e.g., [9]). Despite this fact, and in contrast to SAT solvers, only few CSP solvers use learning, owing to the difficulty of making it cost-effective. Learning in a limited form was present in early CSP solvers, where it was called *nogood learning* [6]. Nogoods are defined as partial assignments that cannot be extended to a full solution. Later *generalized nogoods* [9] (g-nogoods for short) were proposed, which allow *non-assignments* as well, e.g., a g-nogood $(x \nleftarrow 1, y \leftarrow 1)$ means that an assignment in which x is assigned anything but 1 and y is assigned 1 cannot be extended to a solution. This formalism is convenient for representing knowledge obtained by propagators. The g-nogood above, for example, can result from removing 1 from the domain of x, which leads by propagation to removing 1 from the domain of y. G-nogoods may be exponentially stronger than nogoods, as shown in [9].

A more general and succinct representation of learned knowledge is in the form of *signed clauses*. Such clauses are disjunctions of *signed literals*, where a signed literal has the form $v \in D$ or $v \notin D$ (called positive and negative signed literals, respectively), where v is a variable and D is a domain of values. Beckert et al. [3] studied the satisfiability problem of signed CNF, i.e., satisfiability of

a conjunction of signed clauses. They proposed an inference system, based on simplification rules and a rule for binary resolution of signed clauses:

$$\frac{(v \in A \vee X) \quad (v \in B \vee Y)}{(v \in (A \cap B) \vee X \vee Y)} [\text{Signed Resolution}(v)] \tag{1}$$

where X and Y consist of a disjunction of zero or more literals, A and B are sets of values, and v is called the *pivot* variable. Note that in case v is Boolean and A, B are complementary Boolean domains (e.g., $A = \{0\}, B = \{1\}$) then this rule simplifies to the standard resolution rule for propositional clauses that is used in SAT, namely the consequent becomes $(X \vee Y)$.

As we showed in an earlier publication [12], we used this rule in our CSP solver HCSP [12] (short for HaifaCSP)[1], as part of a general learning scheme based on signed clauses. Using a special inference rule for each type of non-clausal constraint, HCSP inferred a signed clause e that *explains* a propagation by that constraint. This means that e is implied by the constraint, but at the same time is strong enough to make the same propagation as the constraint, at the same state. Using such explanations for propagations by non-clausal constraints, and rule (1) for resolving signed clauses, HCSP can generate a signed *conflict clause* via *conflict analysis*. By construction this clause is *asserting* (i.e., it necessarily leads to additional propagation after backtracking). In contrast to the CSP solver EFC [9], which generates a g-nogood *eagerly* for each removed *value*, HCSP generates a signed explanation clause *lazily*, only as part of conflict analysis. Lazy learning of g-nogoods was also implemented on top of MINION [7].

In this article we study a different learning scheme, which is based on inference rules with non-clausal consequents. Our main goal in introducing this scheme is to learn a conflict constraint that is logically stronger and easier to compute than its clausal counterpart. The emphasis is on the first of these goals as it may improve the search itself. To that end, we propose a generic inference rule called *Combine* that for many popular (pairs of) constraints indeed fulfills these two goals. For example, suppose that in a state in which the domains of three variables are defined by $x \in \{2, 6, 10, 14, \ldots, 30\}, y_1 \in \{8, 12, 16, 20\}, y_2 \in \{1, 2, 3, \ldots, 9\}$, the constraint $c_1 \doteq y_1 \leq x$ propagates $x \in \{10, 14, \ldots, 30\}$, which leads to a contradiction with a constraint $c_2 \doteq x \leq y_2$. During conflict-analysis, HCSP now infers from this propagation the constraint

$$x \in [8, 9] \vee [y_1, y_2] \not\subseteq [8, 9] \ ,$$

based on *Combine* (square brackets denote a range). This constraint is both implied by c_1, c_2, and has the form of a disjunction of two constraints, each of which has less variables than the set of input variables. The latter property potentially makes it easier to solve. Instantiations of *Combine* always have this property. For some combinations of rules we do not use *Combine* since the result is too complicated to derive or too computationally expensive to support. In such cases we offer simpler alternatives.

[1] The early version of our solver that was introduced in [12] was called PCS, for Proof-producing Constraint Solver.

Our experimental results prove that indeed the new scheme is better than clausal explanation. For reference, we also compared HCSP to the state-of-the-art CSP solver MISTRAL [8]. Our experiments with thousands of benchmarks from the latest competition (in 2009), as we describe in Sec. 5, show that comparing to MISTRAL, HCSP achieves an overall 25% reduction in fails (time-out or memory-out), 29% reduction in run-time of instances that both engines solved, and 95.9% average reduction in the number of backtracks. Perhaps this drastic reduction in backtracks indicates that the cost of learning strong constraints is mitigated by a better search (MISTRAL itself does not learn constraints).

The rest of the article is structured as follows. The next section covers background material, including the learning framework that we use and clausal explanations [12]. Sections 3 and 4 describe the new set of inference rules, the requirements from them and the proofs that they fulfill these requirements. In Sec. 3 we also explain how we use clausal explanations as a fallback solution when we are unable to infer a general constraint that satisfies the required properties. We conclude in Sec. 5 with empirical evaluation and some proposals for future research.

2 Background

2.1 Essentials of HCSP

The engine of HCSP adopts classical ideas from the CSP and SAT literature. We assume the reader is somewhat familiar with those, and only mention several highlights briefly.

HCSP makes a *decision* (variable ordering) by selecting a variable with the highest ratio of *score* to domain-size, where *score* is calculated similarly to Chaff's VSIDS technique [10].[2] The value is initially chosen to be the minimal value in the domain, and after that according to the last assigned value, a technique that is typically referred to by the name *phase saving* is SAT [11]. It includes *restarts*, *learning* (to be described), and *deletion* of learnt-constraints with low activity.

HCSP supports all the constraint types used in the 2009 CSP competition. It has *precise* propagators for the following types of constraints (where x_i denote variables, b_i Boolean variables, a_i constants, and $\diamond \in \{=, \leq, \geq\}$): $x_0 = x_1, x_0 = -x_1$, $x_0 \diamond abs(x_1)$, $x_0 \diamond x_1 + x_2 + \ldots$ (with wrap on overflow), $x_0 \leq x_1 * x_2$, $x_0 \geq x_1 * x_2$, $x_0 = \min(x_1, x_2, \ldots)$, $x_0 = \max(x_1, x_2, \ldots)$, $x_0 = (b_0 ? x_2 : x_3)$, $x \neq 0 \leftrightarrow y = z$, $x_0 \in Set \leftrightarrow x_1 < x_2$, $x_0 - x_1 \geq a_0$, $(x_0 + a_0 \leq x_1 \vee x_1 + a_1 \geq x_0)$, $x_0 \neq x_1$, AllDifferent$(x_0, x_1, \ldots)$, $[x_0, x_1 + a_0] \subseteq [a_1, a_2]$, $[x_0, x_1 + a_0] \not\subseteq [a_1, a_2]$, $a_0 * x_0 + a_1 * x_1 + ... \geq a_k$, signed-clauses, and a disjunction of any of the above when there are no shared variables. It has *imprecise* propagators for $x_0 = x_1 * x_2$, $x_0 = x_1 / x_2$, $x_0 = x_1 \% x_2$, $x_0 = pow(x_1, x_2)$, and table constraints.

Complex constraints modeled by a language such as XCSP [2] are rewritten into more basic ones.

The rest of this section is focused on the learning mechanism.

[2] This can be seen as a variant of the *dom/wdeg* strategy [4].

2.2 Conflict analysis

An *implication graph* $G(N, E)$ is a directed acyclic graph in which each node $n \in N$ represents a literal (a variable domain) and each edge $c \in E$ represents a constraint. Incoming edges to a node n can only be labeled with the same constraint. Let $(n_1, n), \ldots, (n_k, n)$ be the incoming edges of n, all of which are labeled with a constraint c. This represents the fact that starting with domains $n_1, \ldots, n_k$ the propagator of c inferred the domain in n. The constraint c is called the *antecedent* of n. Each node is also associated with the *decision level* in which the domain reduction occurred.

When an implication graph ends with a conflict (a node labeled with $\perp$), it is called a *conflict graph*. We will follow a convention by which this graph is depicted with the roots at the left and the sink at the right, and the horizontal position of a node indicates the time it occurred. Examples of conflict graphs can be seen in Fig. 1 (the reader is advised to ignore at this stage the distinction between filled and empty nodes in that figure).

HCSP analyzes the conflict graph in order to learn a new constraint, called accordingly a *conflict constraint* (or a conflict *clause* in SAT). A conflict constraint is called *asserting* if there exists a backtrack level in which this constraint necessarily leads to additional propagation. The conflict-analysis function, AnalyzeConflict , indeed computes this level and returns it to the solver, which backtracks accordingly. Alg. 1 shows pseudo-code of AnalyzeConflict as implemented in HCSP. It maintains a set of nodes F, which is initialized to the set of nodes that contradict the input constraint *cc*. In line 4 it performs a *relaxation* of F. Relaxation means that each node in F is 'pushed' to the left as long as the constraint *Cons* remains conflicting. Generally this is possible when domain reductions are redundant, as demonstrated in the following example.

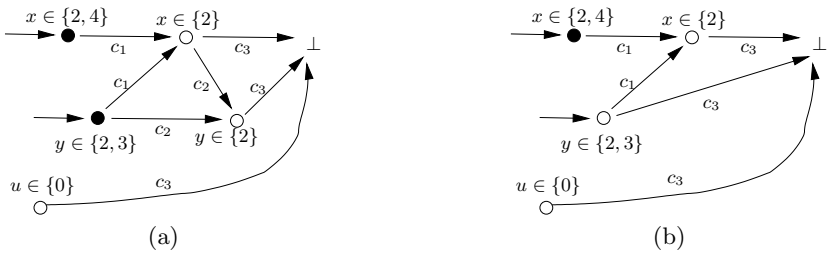


Fig. 1: Part of a conflict graph, based on the constraints $c_1 \doteq y \geq x$, $c_2 \doteq x \geq y$, and $c_3 \doteq x > y + u$. Empty circles represent nodes in the set F. Drawing (a) is before relaxation, and drawing (b) is right after it. Relaxation discovers that the domain reduction by c_2 is not necessary for conflicting the constraint *curr* (c_3 in this case).

Example 1. Consider the constraints

$$c_1 \doteq y \geq x \quad c_2 \doteq x \geq y \quad c_3 \doteq x > y + u \; .$$

and the conflict graph in Fig. 1(a). In Alg. 1, initially $Cons = c_3$, and hence after line 2 $F = \{x \in \{2\}, y \in \{2\}, u \in \{0\}\}$ (those are marked with empty circles). Relaxation in line 4 replaces in F the node $y \in \{2\}$ with the node $y \in \{2, 3\}$, because the new F also contradicts the current constraint $Cons$. Fig. 1(b) shows this. The reason that this is possible is that the domain reduction by c_2 is redundant in the current state, because when $u = 0$, c_3 is capable of removing this value by itself. Such cases appear frequently, because the order in which constraints are processed is not optimal. □

Relaxation is necessary for several reasons:

- Preventing a situation in which the learned clause is still conflicting immediately after backtracking, instead of being asserting,
- In Sec. 4.3 we rely on relaxation in the development of some of the rules.
- Our experiments show that without it many more cases fall back to clausal explanations, because relaxation enables to circumvent them.

Relaxation is a contribution of the current article, although not its focus.

Let us return to the description of Alg. 1. In lines 5–9 AnalyzeConflict gradually updates the constraint $Cons$. It does so by traversing the conflict graph backwards (i.e., going left, from the conflict node towards the decision node) while updating F and the constraint $Cons$ such that the following loop invariants are maintained:

Invar1. *curr* contradicts the domains defined by F, and is able to detect it via propagation (detection is not a given, because not all constraints have a precise propagator, i.e., they are all sound but not all are complete. Bounds consistency is an example of such imprecise propagation).

Invar2. No two nodes in F refer to the same variable.

It should be clear that these invariants are maintained at the entry to the loop, because of the definition of F, $Cons$, and relaxation. Combine and GetNewSet are targeted towards maintaining it as will be evident later. The traversal stops in line 5 once the function stop detects that $Cons$ is asserting, or that it conflicts the domains at decision level 0. In the latter case the function AssertingLevel returns -1 to the solver, which accordingly declares the CSP to be unsatisfiable. In line 8 the current constraint $Cons$ is replaced with a constraint that is inferred from $Cons$ itself and the antecedent constraint of a node in F. The function combine is the main contribution of this article and will be discussed in length in later sections.

Let us now shift our focus to GetNewSet, which updates the set F. Initially it replaces *pivot* with its parents. In case there is more than one node in F representing the same variable, in line 16 the function distinct leaves only the right-most one. The reason that there may be multiple entries of a variable in F is that a parent of *pivot* may represent a variable that already labels a different node in F because of relaxation (line 11) in a previous iteration.

Algorithm 1 AnalyzeConflict receives as input the currently conflicting constraint, learns a new constraint $Cons$ which is asserting (i.e., necessarily leads to further propagation), and returns the backtrack level. Combine, the subject of Sect. 3–4, infers a new constraint. GetNewSet computes the new set of nodes F, as explained in the text.

```
1: function AnalyzeConflict (constraint cc)          ▷ cc = conflicting constraint
2:     F ← the set of nodes contradicting cc;
3:     Cons ← cc;
4:     F ←relax (F, Cons);
5:     while !stop (F,Cons) do      ▷ stop if Cons is asserting or UNSAT detected
6:         pivot ← node of F that was propagated last;
7:         antecedent ← incoming constraint of pivot;
8:         Cons ← Combine (Cons, antecedent, pivot, F);
9:         F ← GetNewSet(F, Cons, pivot);
10:        Remove from F nodes referring to variables not in Cons.
11:        F ← relax (F, Cons);          ▷ Go left as long as F contradicts Cons
12:    Add Cons to the constraints database;
13:    return AssertingLevel (Cons,F); ▷ the backtracking level, or -1 if UNSAT

14: function GetNewSet(node-set F, node pivot)
15:    F ← (F \ {pivot}) ∪ parents of pivot;
16:    F ← distinct (F);          ▷ Chooses right-most node of each variable in F
17:    Return F;
```

2.3 Clausal Explanations

Generic explanations were used in the past (e.g., [9,7]) for learning of g-nogoods. The scheme we describe here uses inference rules specialized for each constraint type, resulting in signed clauses. Such clausal explanations are important in our context both for understanding the alternative mechanism that we used in [12] (we use it as one of the points of reference for comparing the results), and because we still use it as a fallback solution when, e.g., we reach pairs of constraints that we do not directly support in Combine. This technique is also based on AnalyzeConflict , with a difference only in the implementation of Combine.

Let us begin by formally defining the notion of explanation.

Definition 1 (Clausal explanation). *Let $l_1, \ldots, l_n$ be signed literals at the current state (each literal represents the current domain of a variable), and let c be a constraint that propagates the new signed literal l, i.e., $(l_1 \wedge \ldots \wedge l_n \wedge c) \rightarrow l$. Then a clause e is an* explanation *of this propagation if the following two conditions hold:*

$$c \rightarrow e \tag{2}$$

$$(l_1 \wedge \cdots \wedge l_n \wedge e) \rightarrow l\ . \tag{3}$$

Eq. (2) guarantees that the new clause e is logically implied by an existing constraint, hence we do not lose soundness. Eq. (3) guarantees that it is still strong

enough to imply the same literal. It is always possible to derive an explanation from a constraint, regardless of the constraint type [12].

Example 2. The following rule from [12] provides a clausal explanation for an inequality constraint:

$$\frac{x \leq y}{x \in (-\infty, m] \vee y \in [m+1, \infty)} \quad \text{(LE(m))} \tag{4}$$

where m is a parameter instantiating it (the rule is sound for any m). Note that the consequent is a signed clause. Now consider two literals:

$$l_1 \doteq (x \in [1,3]), l_2 \doteq (y \in [0,2])$$

and the constraint

$$c \doteq x \leq y \ ,$$

which implies in the context of l_1, l_2 the literal

$$l \doteq x \in [1,2] \ .$$

Using (4) with $m = \max(y) = 2$ we obtain the explanation

$$e \doteq (x \in (-\infty, 2] \vee y \in [3, \infty)) \ ,$$

and indeed (2) and (3) hold, since $c \to e$ and $(l_1 \wedge l_2 \wedge e) \to l$. In [12] alternatives to choosing $m = \max(y)$ are discussed. □

In [12] we showed how HCSP generates a signed conflict clause with an inference system based on signed resolution (1), that is reminiscent of how SAT solvers use binary resolution. Explanations are used for bridging between non-clausal constraints and a signed clause (as in the example above), and (1) is used for resolving signed clauses.

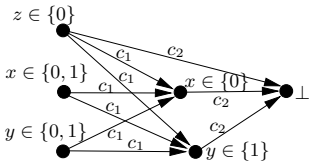
Example 3. The following demonstrates conflict analysis with clausal explanations. In addition to (4), we will use a variant of this rule for strict inequality:

$$\frac{x < y}{x \in (-\infty, m-1] \vee y \in [m+1, \infty)} \quad \text{(L(m))} \tag{5}$$

We will also use the observation that if $c \to e$, then $(l \vee c) \to (l \vee e)$, to handle constraints with disjunctions. Let $D_x = \{0,1\}$, $D_y = \{0,1\}$, $D_z = \{0..100\}$, and

$$c_1 \doteq (z = 9 \vee x < y) \quad c_2 \doteq (z = 10 \vee x \geq y) \ .$$

The conflict graph on the right shows the decision ($D_z = \{0\}$), and then that c_1 propagates $D_x = \{0\}$, $D_y = \{1\}$ in this order, and finally that c_2 detects a conflict. Now $F = \{z \in \{0\}, x \in \{0\}, y \in \{1\}\}$ and $\mathit{pivot} = \{y \in \{1\}\}$. At this point c_2 generates the explanation



$$(z \in \{10\} \vee x \in [1, \infty) \vee y \in (-\infty, 0])$$

based on LE(0) (Eq. (4)), and c_1 generates the explanation

$$(z \in \{9\} \vee y \in [1, \infty) \vee x \in (-\infty, -1])$$

based on L(0) (Eq. (5)). Resolving the two explanations on y yields

$$(z \in \{9, 10\} \vee x \notin \{0\}). \tag{6}$$

Now $pivot = x \in \{0\}$. c_1 explains the propagation of x with the clause $(z \in \{9\} \vee y \in [2, \infty) \vee x \in (-\infty, 0])$, based on L(1). Resolving it with (6) on x yields

$$(z \in \{9, 10\} \vee x \in (\infty, -1] \vee y \in [2, \infty)) \ . \tag{7}$$

Now F is equal to the three nodes on the left. (7) is now asserting, since e.g., at the previous decision level $z \in \{9, 10\}$ is implied. □

3 Non-clausal inference: requirements

In Alg. 1 COMBINE is given the constraints $Cons(x, \bar{y})$ and $antecedent(x, \bar{y})$ with a joint variable x that appears at the node *pivot*, and some set of variables $\bar{y}$, which may or may not be common to both[3]. It outputs a new constraint over $x, \bar{y}$ that is assigned back into $Cons$. In the presentation that follows we will use $c_1(x, \bar{y})$ to denote $Cons(x, \bar{y})$, $c_2(x, \bar{y})$ to denote $antecedent(x, \bar{y})$, and $c^*(x, \bar{y})$ to denote the output constraint. We also define

$$c_{12}(x, \overline{y}) \doteq c_1(x, \overline{y}) \wedge c_2(x, \overline{y}).$$

Typically we will discard the parameters and write c_1, c_2, c^*, c_{12} instead.

Our first requirement from c^* is that it preserves soundness:

$$c_{12} \rightarrow c^* \ . \tag{8}$$

This guarantees that the constraint eventually learned in line 12 is inferred via sound derivations, and hence is guaranteed to be implied by the original CSP.

Let $D'_x, D'_{\overline{y}}$ denote the domains of $x, \overline{y}$ right before the propagation of c_1. Also, let $\vdash_{cp}$ denote the *provability relation* by constraints propagation, i.e., $\phi \vdash_{cp} \psi$ denotes that starting with a set of constraints and domains ϕ, the set of literals ψ is derivable through constraint propagation. Then to preserve *Invar1* (see Sec. 2), our second requirement from c^* is:

$$c^*, D'_x, D'_{\overline{y}} \vdash_{cp} \bot \ . \tag{9}$$

Finally, we aspire to find the strongest c^* that satisfies the above requirements, and which is easy to propagate.

[3] It is of course not necessarily the case that they share all the variables, but the description is simplified if we do not consider the shared and unshared variables separately, without sacrificing correctness.

4 Non-clausal inference: rules and their proofs

Rules R1–R6 in Table 1 are triples $\langle c_1, c_2, c^* \rangle$ that satisfy the two requirements (8) and (9). Rules R7 and R8 satisfy (8) but not necessarily (9). We use them to infer constraints, and then test whether they happen to satisfy (9). In addition, we use the following meta-rule for handling disjunctions:

$$\frac{(A \vee c_1) \quad (B \vee c_2)}{A \vee B \vee c^*} \tag{10}$$

If $\langle c_1, c_2, c^* \rangle$ satisfies (8) and (9), then so does (10).

Proof.

$$\begin{aligned} & (A \vee c_1) \wedge (B \vee c_2) \\ = & (A \wedge B \vee A \wedge c_2 \vee c_1 \wedge B \vee c_1 \wedge c_2) \end{aligned}$$

Clearly

$$A \wedge B \vee A \wedge c_2 \rightarrow A$$

and

$$c_1 \wedge B \rightarrow B$$

and

$$c_1 \wedge c_2 \rightarrow c^*$$

Hence

$$(A \wedge B \vee A \wedge c_2) \vee (c_1 \wedge B) \vee (c_1 \wedge c_2) \rightarrow A \vee B \vee c^*$$

□

Example 4. We now show two examples in which the rules lead to stronger learning than explanation-based learning

- Recall example 3, which yielded the conflict clause (7). Given the same conflict graph but using the meta rule (10) with pivot y, we learn instead $z \in \{9, 10\}$, which is clearly stronger.
- Consider a variant of the example that was described in the introduction: $x \in \{2, 6, 10, 14, \ldots, 30\}, y_1 \in \{8, 12, 16, 20\}, y_2 \in \{1, 2, 3, \ldots, 9\}$, and constraints

  $$c_1 \doteq (z \in \{1\} \vee y_1 \leq x) \quad c_2 \doteq (z \in \{1\} \vee x \leq y_2) \; .$$

  Suppose we make a decision $z \in \{0\}$. Then c_1 propagates $x \in \{10, 14, \ldots, 30\}$ and c_2 detects a conflict. Using rule R2 with $k1 = k2 = 0$, and the meta rule (10) we obtain:

  $$(z \in \{1\} \vee x \in [8, 9] \vee [y_1, y_2] \not\subseteq [8, 9]) \; . \tag{11}$$

  On the other hand if we use explanations, c_2's explanation via LE(9) is $(z \in \{1\} \vee x \in (-\infty, 9] \vee y_2 \in [10, \infty))$, c_1's explanation via LE(7) is

  $$(z \in \{1\} \vee y_1 \in (-\infty, 7] \vee x \in (8, \infty]) \; ,$$

and resolving these explanations on the pivot x yields

$$(z \in \{1\} \vee x \in [8,9] \vee y_1 \in (-\infty, 7] \vee y_2 \in [10, \infty)) \ .$$

This constraint is strictly weaker than (11) because the right disjunct of (11) implies $y_1 \leq y_2$.

□

Most of the entries in the table were developed by instantiating a general inference rule called *Combine* (see below), which satisfies these requirements. In some other cases instantiating it turned out to be too complicated and we found c^* without it. Sec. 4.3 includes proofs for some of these other rules.

	c_1	c_2	c^*
R1	$x \in X_1 \vee A_1(\overline{y})$	$x \in X_2 \vee A_2(\overline{y})$	$x \in (X_1 \cap X_2) \ \vee \ A_1(\overline{y}) \vee A_2(\overline{y})$
R2	$y_1 \leq x - k_1$	$x \leq y_2 - k_2$	$(x \in [k_1 + \min(D'_{y_1}), \max(D'_{y_2}) - k_2]) \ \vee$ $([y_1, y_2 - k_2 - k_1] \not\subseteq [\min(D'_{y_1}), \max(D'_{y_2}) - k_2 - k_1])$
R3	$y_1 \leq x$	$[x, y_2] \not\subseteq [a, b]$	$(a > x \geq \min(D'_{y_1})) \vee$ $([y_1, y_2] \not\subseteq [\min(D'_{y_1}), b])$
R4	$x \leq y_1 - k_1$	$[y_2, x - k_2] \not\subseteq [a, b]$	$(\max(D'_{y_1})\text{-}k_1 \geq x > b + k_2) \vee$ $[y_2, y_1\text{-}k_1\text{-}k_2] \not\subseteq [a, \max(b, \max(D'_{y_1}\text{-}k_1\text{-}k_2))]$
R5	$[y_1, x] \not\subseteq [a_1, b_1]$	$[x, y_2] \not\subseteq [a_2, b_2]$	$(a_2 > x > b_1) \ \vee \ ([y_1, y_2] \not\subseteq [a_1, b_2])$
R6	$[x, y] \not\subseteq [a_1, b_1]$	$[y, x] \not\subseteq [a_2, b_2]$	$(x \in (D'_y \setminus ([a_1, b_1] \cup [a_2, b_2])) \vee$ $y \notin (D'_y \cup [a_1, b_1] \cup [a_2, b_2]))$
R7	$y \leq x + k_1$	$x \leq y + k_2$	$\begin{cases} -k_1 \leq x - y \leq k_2 & \text{if } k_1 + k_2 \geq 0 \\ \bot & \text{otherwise} \end{cases}$
R8	$ax + \sum_{i=1}^{n} a_i y_i \geq k_1$	$-ax + \sum_{i=1}^{n} b_i y_i \geq k_2$	$\sum_{i=1}^{n} (a_i + b_i) x_i \geq k_1 + k_2$

Table 1: Triples $\langle c_1, c_2, c^* \rangle$ that we use for deriving conflict constraints. The top part include rules that satisfy both (8) and (9), whereas the others are only guaranteed to satisfy (8). When using them we *test* if they satisfy (9). Combinations of a pair of linear constraints can be brought to the form expected by R8 if the coefficients of x have opposite signs, via multiplication by a positive constant. Note that the coefficients a_i, b_i may be 0 in R8.

Since not all combinations of rule types are supported, not all propagators are precise (i.e., logically complete) and not all rules are precise (see R7, R8

in the table), then COMBINE uses explanation-based inference (see Sec. 2.3) as a fallback solution. Pseudocode of COMBINE, which is rather self-explanatory, appears in Alg. 2.

Algorithm 2 COMBINE infers a new constraint c^* from c_1, c_2, which satisfies (8) and (9), the requirements listed in Sec. 3.

```
function COMBINE(constraint c1, constraint c2, node pivot, node-set F)
    F' = GETNEWSET (F, pivot);
    if the combination of c1, c2 is supported then
        con = infer(c1,c2, pivot);                    ▷ One of the rules in Table 1.
        if F', con ⊢_cp ⊥ then return con;            ▷ con satisfies Invar1
    e1 ← explain(c1, parents(pivot), pivot);          ▷ Fallback: use explanations.
    e2 ← explain(c2, F, ⊥);
    return resolve(e1,e2,pivot);                      ▷ Signed resolution
```

4.1 A generic inference rule: *Combine*

Let S be some set of values. Then it is not hard to see that the following is a contradiction for any constraint $c(x, \overline{y})$:

$$c(x, \overline{y}) \wedge x \in S \wedge \forall x' \in S.\ \neg c(x', \overline{y})\ , \tag{12}$$

or, equivalently, that the following implication is valid:

$$c(x, \overline{y}) \rightarrow (x \notin S \vee \exists x' \in S.\ c(x', \overline{y}))\ . \tag{13}$$

Let $\mathcal{X}$ denote the set of values of x which have no support in $D'_{\overline{y}}$:

$$\mathcal{X} = \{x' \mid \forall \overline{y'} \in D'_{\overline{y}}\ .\ \neg c_{12}(x', \overline{y'})\}\ . \tag{14}$$

Instantiating (13) with c_{12} for c and with $\mathcal{X}$ for S yields the inference rule that we call *Combine*:

$$\frac{c_{12}(x, \overline{y})}{(x \notin \mathcal{X} \vee \exists x' \in \mathcal{X}.c_{12}(x', \overline{y}))}\ (\mathit{Combine}) \tag{15}$$

Since (15) is just an instantiation of (13), then (15) is clearly sound, and hence (8) is satisfied. To satisfy (9) we first prove logical entailment ($\models$), which is weaker than the requirement of (9) for provability ($\vdash_{cp}$).

Lemma 1. $c^*, D'_x, D'_{\overline{y}} \models \bot$.

Proof. In our case $c^* \doteq (x \notin \mathcal{X} \vee \exists x' \in \mathcal{X}.c_{12}(x', \overline{y}))$. Falsely assume that c^* is satisfied for an assignment of values $a \in D'_x, \overline{b} \in D'_{\overline{y}}$ to $x, \overline{y}$, respectively. Consider the two disjuncts of c^*:

– Suppose $x \notin \mathcal{X}$ is satisfied. Considering the definition of $\mathcal{X}$ in (14), this implies that a is supported in c_{12}, or formally

$$\exists \overline{y'} \in D'_{\overline{y}}.\ c_{12}(a, \overline{y'})\ . \tag{16}$$

Based on *Invar1* we know that $c_{12}(x, \overline{y}), D'_x, D'_{\overline{y}} \models \bot$, and hence $\forall x \in D'_x \neg\exists \overline{y} \in D'_{\overline{y}}.\ c_{12}(x, \overline{y})$, and particularly for $x = a$, $\neg\exists \overline{y} \in D'_{\overline{y}}.\ c_{12}(a, \overline{y})$, which contradicts (16).

– Now suppose $\exists x' \in \mathcal{X}.\ c_{12}(x', \overline{y})$ is satisfied. Expanding $\mathcal{X}$ and substituting $\overline{y}$ with its assignment $\overline{b}$ yields

$$\exists x'.\ \forall \overline{y'} \in D'_{\overline{y}}.\ \neg c_{12}(x', \overline{y'}) \wedge c_{12}(x', \overline{b})\ .$$

Since $\overline{b} \in D'_{\overline{y}}$ and $\neg c_{12}(x', \overline{y'})$ is satisfied for all $\overline{y'} \in D'_{\overline{y}}$, then it is satisfied for $\overline{y'} = \overline{b}$. This implies a contradiction: $\exists x'.\ \neg c_{12}(x', \overline{b}) \wedge c_{12}(x', \overline{b})$.

Hence, $x \in D'_x, \overline{y} \in D'_{\overline{y}}$ falsifies c^*, which completes our proof. □

It is trivial to see that this lemma implies (9) when $\vdash_{cp}$ is precise constraint propagation. When imprecise propagation is involved, e.g., $\vdash_{cp}$ is defined by bounds consistency [5], HCSP checks whether the constraint happens to be conflicting, and if not it falls back to clausal explanation.

The relative strength of *Combine*. Two observations about the strength of *Combine* that we prove below are:

– There is no alternative to $\mathcal{X}$ for replacing S in (13) that makes the resulting constraint stronger, and
– The signed clause that we obtain through the explanation mechanism—see Sec. 2.3—cannot yield a stronger consequent.

Lemma 2. *Consider all possible formulas of the form $\psi(x, \overline{y}) \equiv (x \notin P \vee \varphi(\overline{y}))$, for a given set P. The strongest possible $\varphi(\overline{y})$, which meets all the requirements is $\exists x' \in P.c_{12}(x', \overline{y})$. In other words, for any $\varphi(\overline{y})$ which makes $\psi(x, \overline{y})$ meet the requirements, the following is satisfied:*

$$\exists x' \in P.c_{12}(x', \overline{y}) \models \varphi(\overline{y})$$

Proof. By negation, assume that there is and assignment $\overline{b}$ to $\overline{y}$ which satisfies $\exists x' \in P.c_{12}(x', \overline{b})$ but not $\varphi(\overline{b})$. This means that there is an $a \in P$ which satisfies $c_{12}(a, \overline{b})$ when $\neg\varphi(\overline{b})$. Because $c_{12}(a, \overline{b})$ is satisfied, (8) mandate that $\psi(a, \overline{b})$ should also be satisfied. According to the definition of ψ we conclude that either $a \notin P$ or $\varphi(\overline{b})$ have to be satisfied. But since $\neg\varphi(\overline{b})$ and, as defined, $a \in P$ we conclude that ψ is unsatisfied with $a, \overline{b}$. This conflict the initial assumption. This leads to the conclusion that if $\exists x' \in P.c_{12}(x', \overline{y})$ is satisfied, then $\varphi(\overline{y})$ must also be satisfied.

Note that this lemma refers to P, and not $\mathcal{X}$. This means that it does not rule out the possibility where $\exists x' \in P.c_{12}(x', \overline{y})$ is stronger than $\exists x' \in \mathcal{X}.c_{12}(x', \overline{y})$. It is quite possible that the smaller the set P is and the weaker the literal $x \notin P$ is, the stronger $\exists x' \in P.c_{12}(x', \overline{y})$ becomes.

Lemma 3. *Consider all possible* $\psi(x, \overline{y}) \equiv (x \notin P \vee \exists x' \in P.c_{12}(\overline{y}))$ *which satisfy the requirements.* P *must satisfy*

$$D'_x \subseteq P \ .$$

Proof. By negation assume that $D'_x \not\subseteq P$, i.e., there is a value a such that $a \in D'_x$ and $a \notin P$. Consider how this affects (9). According to (9),

$$x \notin P \vee \exists x' \in P.c_{12}(x', \overline{y}), D'_x, D'_{\overline{y}} \vdash_{cp} \bot \ .$$

Since $a \in D'_x$ then we can replace x with a in the above formula, and get

$$a \notin P \vee \exists x' \in P.c_{12}(x', \overline{y}), D'_{\overline{y}} \vdash_{cp} \bot \ .$$

But since a was defined such that $a \notin P$ then the above formula becomes

$$\texttt{true} \vee \exists x' \in P.c_{12}(x', \overline{y}), D'_{\overline{y}} \vdash_{cp} \bot \ .$$

This basically says $\texttt{true} \vdash_{cp} \bot$, which is impossible. It implies that the assumption that $D'_x \not\subseteq P$ is incorrect. □

Lemma 4. *Consider all possible* $\psi(x, \overline{y}) \equiv (x \notin P \vee \exists x' \in P.c_{12}(\overline{y}))$ *which satisfy the requirements.* P *must satisfy*

$$P \subseteq \mathcal{X} \ .$$

Where $\mathcal{X}$ *was defined above as*

$$\mathcal{X} = \left\{ x' | \forall \overline{y'} \in D'_{\overline{y}}. \left[\neg c_{12}(x', \overline{y'}) \right] \right\} \ .$$

Proof. According to the definition of $\mathcal{X}$, the lemma can be reformulated as

$$\forall a \in P \forall \overline{y'} \in D'_{\overline{y}}. \left[\neg c_{12}(a, \overline{y'}) \right] \ .$$

Assume, by negation, that this is not correct. In other words there are $a \in P$ and $\overline{b} \in D'_{\overline{y}}$ such that $c_{12}(a, \overline{b})$. We will show that this conflicts (9).

Due to (9) we know that

$$x \notin P \vee \exists x' \in P.c_{12}(x', \overline{y}), D'_x, D'_{\overline{y}} \vdash_{cp} \bot \ .$$

Since $\overline{b} \in D'_{\overline{y}}$, the formula above implies

$$x \notin P \vee \exists x' \in P.c_{12}(x', \overline{b}), D'_x \vdash_{cp} \bot \ .$$

Assuming $\vdash_{cp}$ is precise this implies

$$\forall x \in D'_x . \neg \left[x \notin P \vee \exists x' \in P . c_{12}(x', \overline{b}) \right] \ .$$

We now push the negation down, and get

$$\forall x \in D'_x . \left[x \in P \wedge \forall x' \in P . \neg c_{12}(x', \overline{b}) \right] \ .$$

First, we see that this implies $D'_x \subseteq P$. Since x is independent in the formula we conclude that

$$\forall x' \in P . \neg c_{12}(x', \overline{b}) \ .$$

Now we go back to $a \in P$ and $\overline{b} \in D'_{\overline{y}}$ which guarantee $c_{12}(a, \overline{b})$, and combine it with the formula above. This means that we can assign $x' = a$, which leads to $\neg c_{12}(a, \overline{b})$, which conflicts with the guarantee of $c_{12}(a, \overline{b})$. This means that our assumption that the lemma is incorrect was wrong, hence $P \subseteq \mathcal{X}$. □

This lemma implies that the literal $x \notin \mathcal{X}$ is the strongest possible. This does not imply anything regarding the strength of the second part of the formula, i.e., $\exists x' \in \mathcal{X} . [c_{12}(x', \overline{y})]$ may be weakened by strengthening $x \notin P$.

Also note that the previous lemmas bound P to $D'_x \subseteq P \subseteq \mathcal{X}$.

Theorem 1. *There is no alternative $\psi(x, \overline{y})$, different than c^*, which is stronger than c^* with $\mathcal{X}$. In other words*

$$\psi(x, \overline{y}) \not\equiv [x \notin \mathcal{X} \vee \exists x' \in \mathcal{X} . c_{12}(x', \overline{y})]$$

$$\Downarrow$$

$$(\psi(x, \overline{y}) \not\models [x \notin \mathcal{X} \vee \exists x' \in \mathcal{X} . c_{12}(x', \overline{y})]) \ .$$

Proof. Recall that we require that $\psi(x, \overline{y})$ to be of the form $x \notin P \vee \varphi(\overline{y})$. Lemma 2 shows that for a given P, the strongest possible $\varphi(\overline{y})$ is $\exists x' \in P . c_{12}(x', \overline{y})$. This leaves us to prove that

$$x \notin P \vee \exists x' \in P . c_{12}(x', \overline{y})$$

is not stronger than

$$x \notin \mathcal{X} \vee \exists x' \in \mathcal{X} . c_{12}(x', \overline{y}) \ .$$

According to Lemma 4 because P complies with the given requirements then $P \subseteq \mathcal{X}$. If $P = \mathcal{X}$ the two formulas are equivalent and neither are stronger, otherwise $P \subset \mathcal{X}$.

Assume that $P \subset \mathcal{X}$, this means that there is a such that $a \in \mathcal{X}$ and $a \notin$P. As a result, for $x = a$ the literal $x \notin P$ is true and the literal $x \notin \mathcal{X}$ is false. In this situation the P based formula, i.e., $x \notin P \vee \exists x' \in P . c_{12}(x', \overline{y})$ is true, but the $\mathcal{X}$ based formula depends solely on

$$\exists x' \in \mathcal{X} . c_{12}(x', \overline{y}) \ .$$

We look for a case where this formula is falsified when $x = a$. It can be falsified, i.e., not a tautology, since otherwise this would conflict (9). This means

that there is an assignment $\overline{b}$ to $\overline{y}$ such that the formula is falsified. We have found $x = a$ and $y = \overline{b}$ for which the P based formula is satisfied and the $\mathcal{X}$ based formula is falsified. This means that the P based formula is not stronger than the $\mathcal{X}$ based formula.

Because the P based formula is the strongest possible form of $x \notin P \vee \varphi(\overline{y})$, this implies that any $\psi(x, \overline{y})$ that satisfies the requirements is not stronger than

$$x \notin \mathcal{X} \vee \exists x' \in \mathcal{X}.c_{12}(x, \overline{y}) \ .$$

□

Note that this theorem does not say that, with $\mathcal{X}$, the resulting constraint is stronger than any other possibility; it says that no other constraint is stronger. In other words, there need not be a strict ordering of constraints.

4.2 Selected rules based on instantiating *Combine*

We now instantiate *Combine* (15) with several specific constraints of interest. The derivations rely on various properties of the domains before propagation $D'_x, D'_{\overline{y}}$ and right after it $D''_x, D''_{\overline{y}}$. By definition

$$c_1, D'_x, D'_{\overline{y}} \vdash_{cp} (x \in D''_x \wedge y \in D''_{\overline{y}}) \ . \tag{17}$$

We make the following observations about these domains:

1. The domain of x, and possibly domains of variables in $\overline{y}$, are reduced by c_1:

$$D''_x \subset D'_x, \quad D''_{\overline{y}} \subseteq D'_{\overline{y}} \ . \tag{18}$$

2. Owing to *Invar1*, in the context of $D''_x, D''_{\overline{y}}$, c_2 detects a conflict:

$$c_2(x, \overline{y}), D''_x, D''_{\overline{y}} \vdash_{cp} \bot \ . \tag{19}$$

3. c_1 cannot detect a conflict on its own in the context of $D'_x, D'_{\overline{y}}$:

$$c_1, D'_x, D'_{\overline{y}} \nvdash_{cp} \bot \ . \tag{20}$$

We now use these observations when instantiating *Combine*.

Rule R1: $\boldsymbol{c_1 \doteq x \in X_1 \vee A_1(\overline{y})} \qquad \boldsymbol{c_2 \doteq x \in X_2 \vee A_2(\overline{y})}$

A_1 and A_2 are disjunctions of zero or more literals over the variables of $\overline{y}$. Expanding c_{12} in (14) yields

$$\mathcal{X} = \{x' \mid \forall \overline{y'} \in D'_{\overline{y}}. \ (x' \notin X_1 \wedge \neg A_1(\overline{y})) \vee (x' \notin X_2 \wedge \neg A_2(\overline{y}))\} \ .$$

From (17) and (18) we know that $c_1(x, \overline{y}), D'_x, D'_{\overline{y}} \vdash_{cp} x \in D''_x$ and $D'_x \neq D''_x$, which implies that $A_1(\overline{y}), D'_{\overline{y}} \models \bot$, and consequently simplifies the above to

$$\mathcal{X} = \{x' \mid \forall \overline{y'} \in D'_{\overline{y}}. \ x' \notin X_1 \vee (x' \notin X_2 \wedge \neg A_2(\overline{y}))\} \ .$$

Note that the propagation of c_1 in the context of $D'_{\overline{y}}, D'_x$ results in $D''_x = X_1 \cap D'_x$ and $D'_{\overline{y}} = D''_{\overline{y}}$

Since $D'_{\overline{y}} = D''_{\overline{y}}$ and, according to (19), $c_2(x, \overline{y}), D''_x, D''_{\overline{y}} \vdash_{cp} \bot$, then $A_2(\overline{y}), D'_{\overline{y}} \models \bot$. This means that $\forall \overline{y'} \in D'_{\overline{y}}. \neg A_2(\overline{y})$, which simplifies the above formula to

$$\mathcal{X} = \{x' \mid \forall \overline{y'} \in D'_{\overline{y}}.\ x' \notin X_1 \vee x' \notin X_2\} \ .$$

Since the inner part does not depend on $\overline{y}$ the formula is further simplified to

$$\mathcal{X} = \{x' \mid x' \notin X_1 \vee x' \notin X_2\} \ .$$

Using this definition of $\mathcal{X}$ we examine c^*:

$$c^*(x, D'_{\overline{y}}) = (x \in (X_1 \cap X_2) \vee \exists x' \notin (X_1 \cap X_2).\ c_{12}(x', \overline{y})) \ .$$

Let $A' = \exists x' \notin (X_1 \cap X_2).\ c_{12}(x', \overline{y})$. This simplifies the above to

$$c^*(x, D'_{\overline{y}}) = (x \in (X_1 \cap X_2) \vee A') \ . \tag{21}$$

We split the quantifier in A' into three cases:

$$\begin{aligned} A' = \big(&\exists x' \in (X_1 \setminus X_2).\ c_{12}(x', \overline{y}) \vee \\ &\exists x' \in (X_2 \setminus X_1).\ c_{12}(x', \overline{y}) \vee \\ &\exists x' \notin (X_2 \cup X_1).\ c_{12}(x', \overline{y})\big) \quad . \end{aligned}$$

After taking the definitions of c_{12}, c_1, and c_2 into account:

$$\begin{aligned} A' = \big(&\exists x' \in (X_1 \setminus X_2).\ A_2(\overline{y}) \vee \\ &\exists x' \in (X_2 \setminus X_1).\ A_1(\overline{y}) \vee \\ &\exists x' \notin (X_2 \cup X_1).\ A_1(\overline{y}) \wedge A_2(\overline{y})\big) \quad . \end{aligned}$$

Next, we eliminate the $\exists$ quantifier and get

$$\begin{aligned} A' = \big(&(X_1 \setminus X_2 \neq \emptyset \wedge A_2(\overline{y})) \vee \\ &(X_2 \setminus X_1 \neq \emptyset \wedge A_1(\overline{y})) \vee \\ &((X_2 \cup X_1)^C \neq \emptyset \wedge A_1(\overline{y}) \wedge A_2(\overline{y}))\big) \quad . \end{aligned}$$

We simplify this further by showing that $X_1 \setminus X_2 \neq \emptyset$ and $X_2 \setminus X_1 \neq \emptyset$, which leads to $A' = (A_1(\overline{y}) \vee A_2(\overline{y}))$.

According to (17),(18), $(x \in X_1 \vee A_1(\overline{y})), D'_x, D'_{\overline{y}} \models x \in D''_x$, where $D''_x \subset D'_x$, which implies that $A_1(\overline{y}), D'_{\overline{y}} \models \bot$. Similarly $(x \in X_1 \vee A_1(\overline{y})), D''_x, D'_{\overline{y}} \models \bot$ implies that $A_2(\overline{y}), D'_{\overline{y}} \models \bot$. These facts show that A_1 and A_2 are falsified in this context, meaning that we can focus only on $x \in X_1$ and $x \in X_2$ parts of c_1 and c_2 there.

For the following we assume that $\vdash_{cp}$ for signed-clauses is precise, i.e., $\psi \models \phi$ iff $\psi \vdash_{cp} \phi$.

- $X_2 \setminus X_1 \neq \emptyset$. Because $c_2(x, \overline{y}), D'_x, D'_{\overline{y}} \not\vdash_{cp} \bot$ and $\vdash_{cp}$ is assumed to be precise then there is an assignment $a \in D'_x$, $\overline{b} \in D'_{\overline{y}}$ such that $c_2(a, \overline{b})$ is satisfied. Since $A_2(\overline{y}), D'_{\overline{y}} \models \bot$ and $\overline{b} \in D'_{\overline{y}}$ we know that $A_2(\overline{b}) \models \bot$, which implies that $a \in X_2$. Further, since $(c_1(x, \overline{y}) \wedge c_2(x, \overline{y})), D'_x, D'_{\overline{y}} \models \bot$ and $c_2(a, \overline{b})$ is satisfied then $c_1(a, \overline{b}) \models \bot$, i.e., $(a \in X_1 \vee A_1(\overline{b})), D'_{\overline{y}} \models \bot$. This leads to $a \notin X_1$, which together with $c \in X_2$ implies $X_2 \setminus X_1 \neq \emptyset$.
- $X_1 \setminus X_2 \neq \emptyset$. Because $c_1(x, \overline{y}), D'_x, D'_{\overline{y}} \not\models \bot$ and since $A_1(\overline{y}), D'_{\overline{y}} \models \bot$ we know that there is at least one element $a \in D'_x$ such that $a \in X_1$. Again, since $(c_1(a, \overline{y}) \wedge c_2(a, \overline{y})), D'_{\overline{y}} \models \bot$ and $c_1(a, \overline{y}), D'_{\overline{y}} \not\models \bot$ then we know that $c_2(a, \overline{y}), D'_{\overline{y}} \models \bot$. This implies that $a \in X_2 \models \bot$, i.e., $a \notin X_2$. Since $a \notin X_2$ and $a \in X_1$ then it follows that $X_1 \setminus X_2 \neq \emptyset$.

These two facts simplify A' to

$$A' = (A_2(\overline{y}) \vee A_1(\overline{y})) \ ,$$

and correspondingly, according to (21),

$$c^*(x, \overline{y}) = (x \in (X_1 \cap X_2) \vee A_1(\overline{y}) \vee A_2(\overline{y})) \ . \tag{22}$$

Note the equivalence of (22) and the result of signed resolution in (1).

Rule R2: $\boldsymbol{c_1 \doteq y_2 - x \geq k_2}$ $\qquad$ $\boldsymbol{c_2 \doteq x - y_1 \geq k_1}$
Expanding c_{12} in (14) yields

$$\begin{aligned}
\mathcal{X} &= \left\{x' \mid \forall \overline{y'} \in D'_{\overline{y}}.\ [y_2 - x < k_2 \ \vee \ x - y_1 < k_1]\right\} \\
&= \left\{x' \mid \max(D'_{y_2}) - x < k_2 \ \vee \ x - \min(D'_{y_1}) < k_1\right\} \\
&= \left\{x' \mid \max(D'_{y_2}) - k_2 < x \ \vee \ x < k_1 + \min(D'_{y_1})\right\} \ .
\end{aligned}$$

The complement of $\mathcal{X}$ can be written as

$$\mathcal{X}^c = \left[k_1 + \min(D'_{y_1}),\, \max(D'_{y_2}) - k_2\right] \ . \tag{23}$$

Recall (14): $x \notin \mathcal{X} \vee \exists x' \in \mathcal{X}.\ c_{12}(x', \overline{y})$. The right disjunct is equal to:

$$\begin{aligned}
&\exists x'.\ x' \in \mathcal{X} \wedge [y_2 - x' \geq k_2 \wedge x' - y_1 \geq k_1] \\
&= \exists x'.\ x' \in \mathcal{X} \wedge [y_2 - k_2 \geq x' \geq y_1 + k_1] \\
&= \exists x'.\ x' \in \mathcal{X} \wedge x' \in [y_1 + k_1, y_2 - k_2] \ .
\end{aligned} \tag{24}$$

We use (23) to rewrite (24):

$$\exists x'.\ x' \notin \left[k_1 + \min(D'_{y_1}),\, \max(D'_{y_2}) - k_2\right] \wedge x' \in [y_1 + k_1, y_2 - k_2] \ ,$$

which implies

$$\begin{aligned}
&[y_1 + k_1, y_2 - k_2] \not\subseteq \left[k_1 + \min(D'_{y_1}),\, \max(D'_{y_2}) - k_2\right] \\
&= [y_1, y_2 - k_2 - k_1] \not\subseteq \left[\min(D'_{y_1}),\, \max(D'_{y_2}) - k_2 - k_1\right] \ .
\end{aligned}$$

Hence, the rule is

$$\frac{y_2 - x \geq k_2 \qquad x - y_1 \geq k_1}{\begin{array}{c}\big(x \in \big[k_1 + \min(D'_{y_1}),\, \max(D'_{y_2}) - k_2\big] \;\vee \\ [y_1,\, y_2 - k_2 - k_1] \not\subseteq \big[\min(D'_{y_1}),\, \max(D'_{y_2}) - k_2 - k_1\big]\,\big)\end{array}} \tag{25}$$

Rule R4: $\boldsymbol{c_1 \doteq y_1 - x \geq k_1} \qquad \boldsymbol{c_2 \doteq [y_2, x - k_2] \not\subseteq [a_2, b_2]}$

Note that R2 is implied by c_1, c_2. If $y_1 - x \geq k_1$ and $x - y_2 \geq k_2$ conflict D'_x and $D'_{\overline{y}}$ then we can simply use R2. This allows us to concentrate on the case where propagating$[y_2, x - k_2] \not\subseteq [a_2, b_2]$ D'_x and $D'_{\overline{y}}$ removes at least one value from the domain x more than $x - y_2 \geq k_2$. The extra value removed, depicted γ, should satisfy

$$\big(\exists y_2 \in D'_{y_2}.\gamma - y_2 \geq k_2\big) \wedge \forall y_2 \in D'_{y_2}.\, [y_2, \gamma - k_2] \subseteq [a_2, b_2]\ .$$

This is simplified to

$$\gamma - \min(D'_{y_2}) \geq k_2 \wedge [\min(D'_{y_2}), \gamma - k_2] \subseteq [a_2, b_2]$$

From this we conclude, according to the definition of interval inclusion, that

$$\min(D'_{y_2}) \leq \gamma - k_2 \wedge \big(\min(D'_{y_2}) > \gamma - k_2 \vee (a_2 \leq \min(D'_{y_2}) \leq \gamma - k_2 \leq b_2)\big)\ .$$

Since $\min(D'_{y_2}) \leq \gamma - k_2$ and $\min(D'_{y_2}) > \gamma - k_2$ contradict it is possible to eliminate the disjunct $\min(D'_{y_2}) > \gamma - k_2$ and get

$$\min(D'_{y_2}) \leq \gamma - k_2 \wedge a_2 \leq \min(D'_{y_2}) \leq \gamma - k_2 \leq b_2\ .$$

In which we have two copies of $\min(D'_{y_2}) \leq \gamma - k_2$ which one of them can be eliminated

$$a_2 \leq \min(D'_{y_2}) \leq \gamma - k_2 \leq b_2\ .$$

Subsequently

$$a_2 \leq \min(D'_{y_2}) \leq b_2. \tag{26}$$

The next step is to find the value of $\mathcal{X}$.

$$\mathcal{X} = \big\{x' | \forall \overline{y'} \in D'_{\overline{y}}.\neg c_{12}(x', \overline{y'})\big\}\ .$$

we get

$$\mathcal{X} = \big\{x' | \forall \overline{y'} \in D'_{\overline{y}}.\, [y_1 - x < k_1 \vee [y_2, x - k_2] \subseteq [a_2, b_2]]\big\}\ .$$

Considering the properties of $\subseteq$ and of $<$ we replace y_1 with $\max(D'_{y_1})$ and and y_2 with $\min(D'_{y_2})$:

$$\mathcal{X} = \big\{x | \max(D'_{y_1}) - x < k_1 \vee \big[\min(D'_{y_2}), x - k_2\big] \subseteq [a_2, b_2]\big\}\ .$$

When we expand according to the definition of $\subseteq$ we get:

$$\mathcal{X} = \left\{x | \max(D'_{y_1}) - x < k_1 \vee \left(\min(D'_{y_2}) > x - k_2 \vee (\min(D'_{y_2}) \geq a_2 \wedge x - k_2 \leq b_2)\right)\right\} \ .$$

We apply $\min(D'_{y_2}) \geq a_2$, from Eq.(26), on the above equation

$$\mathcal{X} = \left\{x | \max(D'_{y_1}) - x < k_1 \vee \left(\min(D'_{y_2}) > x - k_2 \vee x - k_2 \leq b_2\right)\right\} \ .$$

We also apply $b_2 \geq \min(D'_{y_2})$, from Eq.(26), on the above equation

$$\mathcal{X} = \left\{x | \max(D'_{y_1}) - x < k_1 \vee x - k_2 \leq b_2\right\} \ .$$

Normalizing x gives

$$\mathcal{X} = \left\{x | \max(D'_{y_1}) - k_1 < x \vee x \leq b_2 + k_2\right\} \ .$$

In terms of intervals, this can be written as

$$\mathcal{X} = (b_2 + k_2, \max(D'_{y_1}) - k_1]^C \quad ,$$

and if we assume that all values are integers then

$$\mathcal{X} = [b_2 + k_2 + 1, \max(D'_{y_1}) - k_1]^C \ .$$

$$c^* \doteq x \notin \mathcal{X} \vee \exists x' \in \mathcal{X}. \, (y_1 - x' \geq k_1 \wedge [y_2, x' - k_2] \not\subseteq [a_2, b_2]) \ .$$

We explore the right disjunct

$$\exists x' \in \mathcal{X}. \, (y_1 - k_1 \geq x' \wedge [y_2, x' - k_2] \not\subseteq [a_2, b_2]) \ .$$

Applying $\mathcal{X}$ we get

$$\exists x'. \left[\max(D'_{y_1}) - k_1 < x' \wedge y_1 - x' \geq k_1 \wedge [y_2, x' - k_2] \not\subseteq [a_2, b_2]\right] \vee$$
$$\exists x'. \, [x' \leq b_2 + k_2 \wedge y_1 - x' \geq k_1 \wedge [y_2, x' - k_2] \not\subseteq [a_2, b_2]] \ .$$

Expanding the definition of $\not\subseteq$ gives

$$\exists x'. \left[\max(D'_{y_1}) - k_1 < x' \wedge y_1 - x' \geq k_1 \wedge y_2 \leq x' - k_2 \wedge y_2 < a_2\right] \vee$$
$$\exists x'. \left[\max(D'_{y_1}) - k_1 < x' \wedge y_1 - x' \geq k_1 \wedge y_2 \leq x' - k_2 \wedge x' - k_2 > b_2\right] \vee$$
$$\exists x'. \, [x' \leq b_2 + k_2 \wedge y_1 - x' \geq k_1 \wedge y_2 \leq x' - k_2 \wedge y_2 < a_2] \vee$$
$$\exists x'. \, [x' \leq b_2 + k_2 \wedge y_1 - x' \geq k_1 \wedge y_2 \leq x' - k_2 \wedge x' - k_2 > b_2] \ .$$

Now, $x' \leq b_2 + k_2$ conflicts $x' - k_2 > b_2$ in the last disjunct, which implies:

$$\exists x'. \left[\max(D'_{y_1}) - k_1 < x' \wedge y_1 - x' \geq k_1 \wedge y_2 \leq x' - k_2 \wedge y_2 < a_2\right] \vee$$
$$\exists x'. \left[\max(D'_{y_1}) - k_1 < x' \wedge y_1 - x' \geq k_1 \wedge y_2 \leq x' - k_2 \wedge x' - k_2 > b_2\right] \vee$$
$$\exists x'. \, [x' \leq b_2 + k_2 \wedge y_1 - x' \geq k_1 \wedge y_2 \leq x' - k_2 \wedge y_2 < a_2] \ .$$

We move x' and replace $<$ with $\leq$ as a preparation to eliminate x':

$$\exists x'.\, \left[\max(D'_{y_1}) - k_1 + 1 \leq x' \wedge \underline{y_1 - k_1 \geq x'} \wedge y_2 + k_2 \leq x' \wedge y_2 < a_2\right] \vee$$
$$\exists x'.\, \left[\max(D'_{y_1}) - k_1 + 1 \leq x' \wedge \underline{y_1 - k_1 \geq x'} \wedge y_2 + k_2 \leq x' \wedge x' \geq b_2 + k_2 + 1\right] \vee$$
$$\exists x'.\, \left[x' \leq b_2 + k_2 \wedge y_1 - k_1 \geq x' \wedge \underline{y_2 + k_2 \leq x'} \wedge y_2 < a_2\right] \ .$$

We eliminate x' in:

$$\left[\max(D'_{y_1}) - k_1 + 1 \leq y_1 - k_1 \wedge y_2 + k_2 \leq y_1 - k_1 \wedge y_2 < a_2\right] \vee$$
$$\left[\max(D'_{y_1}) - k_1 + 1 \leq y_1 - k_1 \wedge y_2 + k_2 \leq y_1 - k_1 \wedge y_1 - k_1 \geq b_2 + k_2 + 1\right] \vee$$
$$\left[y_2 + k_2 \leq b_2 + k_2 \wedge y_1 - k_1 \geq y_2 + k_2 \wedge y_2 < a_2\right] \ .$$

After some normalization:

$$\left[\max(D'_{y_1}) + 1 \leq y_1 \wedge y_2 + k_2 \leq y_1 - k_1 \wedge y_2 < a_2\right] \vee$$
$$\left[\max(D'_{y_1}) + 1 \leq y_1 \wedge y_2 + k_2 \leq y_1 - k_1 \wedge y_1 - k_1 \geq b_2 + k_2 + 1\right] \vee$$
$$\left[y_2 \leq b_2 \wedge y_1 - k_1 \geq y_2 + k_2 \wedge y_2 < a_2\right] \ .$$

In the last disjunct, since $y_2 < a_2$ and we know that $a_2 \leq b_2$ then $y_2 \leq b$ is redundant and the last disjunct becomes

$$y_1 - k_1 \geq y_2 + k_2 \wedge y_2 < a_2 \ .$$

This subsumes the first disjunct:

$$\max(D'_{y_1}) + 1 \leq y_1 \wedge y_2 + k_2 \leq y_1 - k_1 \wedge y_2 < a_2$$

So we are left with

$$\left[\max(D'_{y_1}) + 1 \leq y_1 \wedge \underline{y_2 + k_2 \leq y_1 - k_1} \wedge y_1 - k_1 \geq b_2 + k_2 + 1\right] \vee$$
$$\left[\underline{y_1 - k_1 \geq y_2 + k_2} \wedge y_2 < a_2\right] \ .$$

We now define $k_1 + k_2 = k^*$ which leaves us with:

$$\left[\max(D'_{y_1}) + 1 \leq y_1 \wedge \underline{y_2 + k^* \leq y_1} \wedge y_1 \geq b_2 + k^* + 1\right] \vee$$
$$\left[\underline{y_1 \geq y_2 + k^*} \wedge y_2 < a_2\right] \ .$$

Combining conjunctions:

$$\left[\underline{y_2 + k^* \leq y_1} \wedge y_1 \geq \max(b_2 + k^* + 1, \max(D'_{y_1}) + 1)\right] \vee$$
$$\left[\underline{y_1 \geq y_2 + k^*} \wedge y_2 < a_2\right] \ .$$

This can be written as a interval constraint:

$$[y_2 + k^*, y_1] \not\subseteq \left[a_2 + k^*, \max(b_2 + k^*, \max(D'_{y_1}))\right]$$

We can subtract k^* from all sides and get:

$$[y_2, y_1 - k_1 - k_2] \not\subseteq \left[a_2, \max(b_2, \max(D'_{y_1}) - k_1 - k_2)\right]$$

The rule then becomes

$$\frac{y_1 - x \geq k_1 \qquad [y_2, x - k_2] \not\subseteq [a_2, b_2]}{\begin{array}{l} b_2 + k_2 < x \leq \max(D'_{y_1}) - k_1 \vee \\ [y_2, y_1 - k_1 - k_2] \not\subseteq \left[a_2, \max(b_2, \max(D'_{y_1}) - k_1 - k_2)\right] \end{array}}$$

Rule R5: $\boldsymbol{c_1 \doteq [y_1, x] \not\subseteq [a_1, b_1]} \qquad \boldsymbol{c_2 \doteq [x, y_2] \not\subseteq [a_2, b_2]}$

We demand that at the point of conflict, replacing either one of the constraints with a plain $\leq$ will not detect a conflict. If replacing with $\leq$ detects the conflict then we simply employ R2. This means that $[y_1, x] \not\subseteq [a_1, b_1]$ removes at least one value from D'_x more than $y_1 \leq x$. The extra value removed γ should satisfy

$$\min(D'_{y_1}) \leq \gamma \wedge \forall y_1 \in D'_{y_1}.[y_1, \gamma] \subseteq [a_1, b_1] \ .$$

This means that

$$\min(D'_{y_1}) \leq \gamma \wedge [\min(D'_{y_1}), \gamma] \subseteq [a_1, b_1] \ .$$

From this we conclude that

$$a_1 \leq \min(D'_{y_1}) \leq \gamma \leq b_1 \ .$$

Similarly from the second constraint we conclude that

$$a_2 \leq \max(D'_{y_2}) \leq b_2 \ .$$

We require that neither constraints detect a conflict by themselves with D'_{y_1}, D'_{y_2}, and D'_x, but when propagated consecutively they reach a conflict. This means that there is at least one value $\gamma \in D'_x$ such that

$$\begin{array}{c} \exists y_1 \in D'_{y_1}.[y_1, \gamma] \not\subseteq [a_1, b_1] \wedge \\ \forall y_2 \in D'_{y_2}.[\gamma, y_2] \subseteq [a_2, b_2] \end{array}$$

This can be simplified to

$$[\min(D'_{y_1}), \gamma] \not\subseteq [a_1, b_1] \wedge [\gamma, \max(D'_{y_2})] \subseteq [a_2, b_2] \ .$$

If such γ exists then it can be equal to $\max(D'_x)$ such that

$$[\min(D'_{y_1}), \max(D'_x)] \not\subseteq [a_1, b_1] \wedge [\max(D'_x), \max(D'_{y_2})] \subseteq [a_2, b_2] \ .$$

To summarize, we have concluded that

$$a_1 \leq \min(D'_{y_1}) \leq b_1 \wedge [\min(D'_{y_1}), \max(D'_x)] \not\subseteq [a_1, b_1] \wedge$$
$$[\max(D'_x), \max(D'_{y_2})] \subseteq [a_2, b_2] \wedge a_2 \leq \max(D'_{y_2}) \leq b_2 \ .$$

The interval expressions can be expanded such that

$$a_1 \leq \min(D'_{y_1}) \leq b_1 \wedge \min(D'_{y_1}) \leq \max(D'_x) \wedge$$
$$(\min(D'_{y_1}) < a_1 \vee \max(D'_x) > b_1) \wedge (\max(D'_x) > \max(D'_{y_2})$$
$$\vee (\max(D'_x) \geq a_2 \wedge \max(D'_{y_2}) \leq b_2)) \wedge a_2 \leq \max(D'_{y_2}) \leq b_2 \ .$$

This is simplified to

$$a_1 \leq \min(D'_{y_1}) \leq b_1 < \max(D'_x) \wedge \max(D'_x) \geq a_2 \wedge a_2 \leq \max(D'_{y_2}) \leq b_2 \ .$$

We will depict this as

$$\psi \triangleq (a_1 \leq \min(D'_{y_1}) \leq b_1 < \max(D'_x) \wedge \max(D'_x) \geq a_2 \wedge a_2 \leq \max(D'_{y_2}) \leq b_2) \ . \tag{27}$$

We require that the resulting constraint be of the form

$$x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [a^*, b^*] \quad ,$$

and meet the soundness and completeness requirements. First we look for $\mathcal{X}$which is defined by

$$\mathcal{X} = \left\{ x' | \forall y_1 \in D'_{y_1} \forall y_2 \in D'_{y_2} . \, [[y_1, x'] \subseteq [a_1, b_1] \vee [x', y_2] \subseteq [a_2, b_2]] \right\} \ .$$

We expand the interval operators to

$$\mathcal{X} = \{ x' | \forall y_1 \in D'_{y_1} \forall y_2 \in D'_{y_2} .$$
$$y_1 > x' \vee (a_1 \leq y_1 \wedge x' \leq b_1) \vee x' > y_2 \vee (a_2 \leq x' \wedge y_2 \leq b_2) \} \ .$$

Since y_1 is bounded only from below and y_2 only from above then we can safely rewrite this expression for the worse cases of y_1 and y_2 which are $\min(D'_{y_1})$ and $\max(D'_{y_2})$:

$$\mathcal{X} = \{ x' | \min(D'_{y_1}) > x' \vee (a_1 \leq \min(D'_{y_1}) \wedge x' \leq b_1) \vee$$
$$x' > \max(D'_{y_2}) \vee (a_2 \leq x' \wedge \max(D'_{y_2}) \leq b_2) \} \ .$$

From the initial assumptions, as expressed by Equation (27), we know that $a_1 \leq \min(D'_{y_1})$ and $\max(D'_{y_2}) \leq b_2$ are true. From, this we conclude that that $(a_1 \leq \min(D'_{y_1}) \wedge x' \leq b_1)$ can be simplified to $x' \leq b_1$ and that $(a_2 \leq x' \wedge \max(D'_{y_2}) \leq b_2)$ can be simplified to $a \leq x'$. This yields

$$\mathcal{X} = \left\{ x' | \min(D'_{y_1}) > x' \vee x' \leq b_1 \vee x' > \max(D'_{y_2}) \vee a_2 \leq x' \right\} \ .$$

Equation (27) also states that $\min(D'_{y_1}) \leq b_1$ and $a_2 \leq \max(D'_{y_2})$. This makes both $\min(D'_{y_1}) > x'$ and $x' > \max(D'_{y_2})$ redundant in the expression above. So the expression becomes

$$\mathcal{X} = \{x' | x' \leq b_1 \vee a_2 \leq x'\} \ .$$

$$X = (-\infty, b_1] \cup [a_2, \infty) \ .$$

We will look for the values of a^* and b^* which produce the strongest $[y_1, y_2] \not\subseteq [a^*, b^*]$ part which meets our requirements. This can be achieved by finding the smallest a^* and the biggest possible b^*. First we require that the constraint is sound, i.e., derivable from the two original constraints:

$$\forall x \forall y_1 \forall y_2 . \, [\psi \wedge ([y_1, x] \not\subseteq [a_1, b_1] \wedge [x, y_2] \not\subseteq [a_2, b_2]) \rightarrow (x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [a^*, b^*])] \ .$$

Where ψ is defined in Equation (27). At first glance it seems that ψ does not effect the expression. We will omit ψ as it will, at most, strengthen the expression. Assume by negation that this is not so, i.e., there are x, y_1, y_2 that violate it. This means that

$$[y_1, x] \not\subseteq [a_1, b_1] \wedge [x, y_2] \not\subseteq [a_2, b_2] \wedge (x \leq b_1 \vee a_2 \leq x) \wedge [y_1, y_2] \subseteq [a^*, b^*] \ .$$

From $[y_1, x] \not\subseteq [a_1, b_1] \wedge [x, y_2] \not\subseteq [a_2, b_2]$ we conclude that $y_1 \leq x \leq y_2$. This means that $[y_1, y_2] \subseteq [a^*, b^*]$ can be replaced with $y_1 \geq a^* \wedge y_2 \leq b^*$:

$$[y_1, x] \not\subseteq [a_1, b_1] \wedge [x, y_2] \not\subseteq [a_2, b_2] \wedge (x \leq b_1 \vee a_2 \leq x) \wedge y_1 \geq a^* \wedge y_2 \leq b^* \ .$$

The best case scenario is when $y_1 = a^*$ and $y_2 = b^*$ as they maximize the intervals $[y_1, x]$ and $[x, y_2]$. This gives

$$[a^*, x] \not\subseteq [a_1, b_1] \wedge [x, b^*] \not\subseteq [a_2, b_2] \wedge (x \leq b_1 \vee a_2 \leq x) \ .$$

Consider the two possibilities for x, as expressed in the above expression in $x \leq b_1 \vee a_2 \leq x$. If $x \leq b_1$ then $[a^*, x] \not\subseteq [a_1, b_1]$ implies $a^* < a_1$ and if $x \geq a_2$ then $[x, b^*] \not\subseteq [a_2, b_2]$ implies $b^* > b_2$. In order to violate this, i.e., there will be no x to satisfy this equation, we require that $a^* \geq a_1 \wedge b_2 \leq b^*$. So it seems that the strongest constraint that matches the requirements is

$$x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [a_1, b_2] \ .$$

When

$$\mathcal{X} = \{x' | x' \leq b_1 \vee a_2 \leq x'\} \ .$$

– Soundness: We need to check if

$$\begin{gathered} \big(a_1 \leq \min(D'_{y_1}) \leq b_1 \wedge a_2 \leq \max(D'_{y_2}) \leq b_2 \wedge [y_1, x] \not\subseteq [a_1, b_1] \wedge [x, y_2] \not\subseteq [a_2, b_2]\big) \\ \models \\ (x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [a_1, b_2]) \end{gathered}$$

By negation assume that this is not so, i.e., there are x, y_1, y_2 such that

$$a_1 \leq \min(D'_{y_1}) \leq b_1 \wedge a_2 \leq \max(D'_{y_2}) \leq b_2 \wedge [y_1, x] \not\subseteq [a_1, b_1] \wedge [x, y_2] \not\subseteq [a_2, b_2] \wedge x \in \mathcal{X} \wedge [y_1, y_2] \subseteq [a_1, b_2] \ .$$

We expand the definition of $\mathcal{X}$ and get

$$a_1 \leq \min(D'_{y_1}) \leq b_1 \wedge a_2 \leq \max(D'_{y_2}) \leq b_2 \wedge [y_1, x] \not\subseteq [a_1, b_1] \wedge [x, y_2] \not\subseteq [a_2, b_2] \wedge (b_1 \geq x \vee x \geq a_2) \wedge [y_1, y_2] \subseteq [a_1, b_2] \ .$$

If $x \leq b_1$ then $[y_1, x] \not\subseteq [a_1, b_1]$ implies $y_1 < a_1$ which conflicts $[y_1, y_2] \subseteq [a_1, b_2]$. Otherwise, $a_2 \geq x$ and $[x, y_2] \not\subseteq [a_2, b_2]$ implies $y_2 > b_2$ which conflicts $[y_1, y_2] \subseteq [a_1, b_2]$. This means that there exists no x that satisfies the above expression, which conflicts our assumption that the new constraint is not derived from the original two interval constraints.

– Completeness: We need to check that

$$\psi \rightarrow \left(x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [a_1, b_2], D'_x, D'_{y_1} D'_{y_2} \vdash \emptyset\right) \ .$$

Where ψ is an expression defined by Equation (27). By negation we assume that

$$\psi \rightarrow \exists x \in D'_x \exists y_1 \in D'_{y_1} \exists y_2 \in D'_{y_2}. [x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [a_1, b_2]] \ .$$

The part of x can be simplified such that

$$\psi \rightarrow \exists y_1 \in D'_{y_1} \exists y_2 \in D'_{y_2}. [D'_x \setminus \mathcal{X} \neq \emptyset \vee [y_1, y_2] \not\subseteq [a_1, b_2]] \ .$$

But according to the construction of $\mathcal{X}$ we know that $D'_x \setminus \mathcal{X} = \emptyset$, so the expression is further simplified to

$$\psi \rightarrow \exists y_1 \in D'_{y_1} \exists y_2 \in D'_{y_2}. [y_1, y_2] \not\subseteq [a_1, b_2] \ .$$

We expand it according to the definition of interval constraints:

$$\psi \rightarrow \exists y_1 \in D'_{y_1} \exists y_2 \in D'_{y_2}. [y_1 \leq y_2 \wedge (y_1 < a_1 \vee b_2 < y_2)] \ .$$

Since y_1 is bounded only from above and y_2 is bounded from below then we can replace y_1 with $\min(D'_{y_1})$ and y_2 with $\max(D'_{y_2})$:

$$\psi \rightarrow \left[\min(D'_{y_1}) \leq \max(D'_{y_2}) \wedge (\min(D'_{y_1}) < a_1 \vee b_2 < \max(D'_{y_2}))\right] \ .$$

From the definition of ψ we know that $\min(D'_{y_1}) < a_1$ and $b_2 < \max(D'_{y_2})$ are false, so $(\min(D'_{y_1}) < a_1 \vee b_2 < \max(D'_{y_2}))$ is false and the expression above is falsified. This means that our assumption was wrong, i.e., our completeness requirement is not violated.

To summarize this part, the Combination rule is

$$\frac{[y_1, x] \not\subseteq [a_1, b_1] \qquad [x, y_2] \not\subseteq [a_2, b_2] \qquad \psi}{a_2 > x > b_1 \vee [y_1, y_2] \not\subseteq [a_1, b_2]} \ . \tag{28}$$

Rule R6: $\boldsymbol{c_1 \doteq [x, y] \not\sqsubseteq [a_1, b_1]} \qquad \boldsymbol{c_2 \doteq [y, x] \not\sqsubseteq [a_2, b_2]}$
We have $c_1 \wedge c_2 \rightarrow x = y$, because otherwise, e.g., if $x < y$, then c_2 is trivially false. Since $x = y$ then their joint value cannot be contained in either of the ranges $[a_1, b_1], [a_2, b_2]$. Hence,

$$c_{12} = (x = y) \wedge x \notin [a_1, b_1] \wedge x \notin [a_2, b_2] \,, \tag{29}$$

which implies that

$$\begin{aligned} \mathcal{X} &= \{x' \mid \forall y \in D'_y.\ x \neq y \vee x \in [a_1, b_1] \vee x \in [a_2, b_2]]\} \\ &= D'^C_y \cup [a_1, b_1] \cup [a_2, b_2] \,. \end{aligned}$$

The consequent of *Combine* is

$$\begin{aligned} c^* &= (x \notin \mathcal{X} \vee \exists x \in \mathcal{X}.\ x = y \wedge x \notin [a_1, b_1] \wedge x \notin [a_2, b_2]) \\ &= (x \notin \mathcal{X} \vee (y \in \mathcal{X} \wedge y \notin [a_1, b_1] \wedge y \notin [a_2, b_2])) \\ &= (x \notin \mathcal{X} \vee ((y \in [a_1, b_1] \vee y \in [a_2, b_2] \vee y \notin D'_y) \wedge y \notin [a_1, b_1] \wedge y \notin [a_2, b_2])) \\ &= (x \notin \mathcal{X} \vee (y \notin D'_y \wedge y \notin [a_1, b_1] \wedge y \notin [a_2, b_2])) \\ &= (x \in (D'_y \setminus ([a_1, b_1] \cup [a_2, b_2])) \vee y \notin (D'_y \cup [a_1, b_1] \cup [a_2, b_2])) \,. \end{aligned}$$

Hence, the resulting rule in this case is

$$\frac{[x, y] \not\sqsubseteq [a_1, b_1] \qquad [y, x] \not\sqsubseteq [a_2, b_2]}{(x \in (D'_y \setminus ([a_1, b_1] \cup [a_2, b_2])) \vee y \notin (D'_y \cup [a_1, b_1] \cup [a_2, b_2]))} \tag{30}$$

4.3 Selected rules not based on *Combine*

Rule R3: $\boldsymbol{c_1 \doteq y_1 \leq x} \qquad \boldsymbol{c_2 \doteq [x, y_2] \not\sqsubseteq [a, b]}$
We assume that at the point of conflict, replacing c_2 with $x \leq y_2$ makes c_{12} too weak to detect the conflict. Otherwise we simply use rule R2. Based on this assumption, which we denote by ψ, we now develop $\mathcal{X}$. ψ means that $[x, y_2] \not\sqsubseteq [a, b]$ removes at least one value from D'_x more than $x \leq y_2$. The extra value removed α should satisfy

$$\alpha \leq \max(D'_{y_2}) \wedge \forall y_2 \in D'_{y_2}.[\alpha, y_2] \subseteq [a, b] \,.$$

This means that

$$\alpha \leq \max(D'_{y_2}) \wedge [\alpha, \max(D'_{y_2})] \subseteq [a, b] \,.$$

From this we conclude that

$$a \leq \alpha \leq \max(D'_{y_2}) \leq b \,.$$

We require that neither constraints detect a conflict by themselves with D'_{y_1}, D'_{y_2}, and D'_x, but when propagated consecutively they reach a conflict. This means that there is at least one value $\alpha \in D'_x$ such that

$$\begin{aligned} &\exists y_1 \in D'_{y_1}.y_1 \leq \alpha \wedge \\ &\forall y_2 \in D'_{y_2}.[\alpha, y_2] \subseteq [a, b] \end{aligned}$$

This can be simplified to

$$\min(D'_{y_1}) \leq \alpha \wedge [\alpha, \max(D'_{y_2})] \subseteq [a, b] \ .$$

If such α exists then it can be equal to $\max(D'_x)$ such that

$$\min(D'_{y_1}) \leq \max(D'_x) \wedge [\max(D'_x), \max(D'_{y_2})] \subseteq [a, b] \ .$$

To summarize, we have concluded that

$$a \leq \max(D'_{y_2}) \leq b \wedge \min(D'_{y_1}) \leq \max(D'_x) \wedge [\max(D'_x), \max(D'_{y_2})] \subseteq [a, b] \ .$$

The interval expression can be expanded such that

$$\begin{aligned} &a \leq \max(D'_{y_2}) \leq b \wedge \min(D'_{y_1}) \leq \max(D'_x) \wedge (\max(D'_x) > \max(D'_{y_2}) \vee \\ &(a \leq \max(D'_x) \wedge \max(D'_{y_2}) \leq b)) \ . \end{aligned}$$

This is simplified to

$$a \leq \max(D'_{y_2}) \leq b \wedge \min(D'_{y_1}) \leq \max(D'_x) \wedge (\max(D'_x) > \max(D'_{y_2}) \vee a \leq \max(D'_x)) \ .$$

Since $a \leq \max(D'_{y_2})$ then if $\max(D'_x) > \max(D'_{y_2})$ is satisfied then $\max(D'_x) > \max(D'_{y_2}) \geq a$, i.e., $D'_x \geq a$, is implied. This means that $\max(D'_x) > \max(D'_{y_2})$ is redundant in the above expression, generating:

$$a \leq \max(D'_{y_2}) \leq b \wedge \min(D'_{y_1}) \leq \max(D'_x) \wedge a \leq \max(D'_x) \ .$$

We will depict this as

$$\psi' \triangleq (a \leq \max(D'_{y_2}) \leq b \wedge \min(D'_{y_1}) \leq \max(D'_x) \wedge a \leq \max(D'_x)) \ . \qquad (31)$$

We require that the resulting constraint be of the form

$$x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [a^*, b^*] \quad ,$$

and meet the soundness and completeness requirements. First we look for $\mathcal{X}$ which is defined by

$$\mathcal{X} = \left\{ x' | \forall y_1 \in D'_{y_1} \forall y_2 \in D'_{y_2} . [y_1 > x' \vee [x', y_2] \subseteq [a, b]] \right\} \ .$$

We expand the interval operator to

$$\mathcal{X} = \left\{ x' | \forall y_1 \in D'_{y_1} \forall y_2 \in D'_{y_2} . [y_1 > x' \vee x' > y_2 \vee (a \leq x' \wedge y_2 \leq b)] \right\} \ .$$

Since y_1 is bounded only from below and y_2 only from above then we can safely rewrite this expression for the worse cases of y_1 and y_2 which are $\min(D'_{y_1})$ and $\max(D'_{y_2})$:

$$\mathcal{X} = \left\{ x' | \min(D'_{y_1}) > x' \vee x' > \max(D'_{y_2}) \vee (a \leq x' \wedge \max(D'_{y_2}) \leq b) \right\} \ .$$

From the initial assumptions, as expressed by Equation (31), we know that $\max(D'_{y_2}) \leq b$ is true. From, this we conclude that that $(a \leq x' \wedge \max(D'_{y_2}) \leq b)$ can be simplified to $a \leq x'$. This yields

$$\mathcal{X} = \left\{x' | \min(D'_{y_1}) > x' \vee x' > \max(D'_{y_2}) \vee a \leq x'\right\} .$$

Equation (31) also states that $a \leq \max(D'_{y_2})$ which makes $x' > \max(D'_{y_2})$ redundant in $x' > \max(D'_{y_2}) \vee a \leq x$. So the expression becomes

$$\mathcal{X} = \{x' \mid x' < \min(D'_{y_1}) \vee a \leq x'\} . \tag{32}$$

We propose the following consequent:

$$c^* \doteq x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [\min(D'_{y_1}), b] \tag{33}$$

$$= a > x \geq \min(D'_{y_1}) \vee [y_1, y_2] \not\subseteq [\min(D'_{y_1}), b] . \tag{34}$$

Note that c^* still follows our general pattern, by which the pivot is separated and not referred-to by the other disjunct. Since we cannot rely on the correctness of the general rule, we now prove that (34) satisfies (8) and (9):

- Eq. (8): Falsely assume the contrary, i.e., there are x, y_1, y_2 such that

  $$a \leq \max(D'_{y_2}) \leq b \wedge \min(D'_{y_1}) \leq \max(D'_x) \wedge a \leq \max(D'_x) \wedge y_1 \leq x$$
  $$\wedge [x, y_2] \not\subseteq [a, b] \wedge x \in \mathcal{X} \wedge [y_1, y_2] \subseteq [\min(D'_{y_1}), b] .$$

  Expanding $\mathcal{X}$ yields

  $$a \leq \max(D'_{y_2}) \leq b \wedge \min(D'_{y_1}) \leq \max(D'_x) \wedge a \leq \max(D'_x) \wedge y_1 \leq x$$
  $$\wedge [x, y_2] \not\subseteq [a, b] \wedge (x < \min(D'_{y_1}) \vee a \leq x) \wedge [y_1, y_2] \subseteq [\min(D'_{y_1}), b] .$$

  If $x < \min(D'_{y_1})$ then $y_1 \leq x$ implies $y_1 < \min(D'_{y_1})$ which conflicts $[y_1, y_2] \subseteq [\min(D'_{y_1}), b]$. Otherwise, $x \geq a$ and $[x, y_2] \not\subseteq [a, b]$ implies $y_2 > b$ which conflicts $[y_1, y_2] \subseteq [\min(D'_{y_1}), b]$.
- Eq. (9): We need to check that

  $$\psi' \rightarrow (x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [\min(D'_{y_1}), b], D'_x, D'_{y_1} D'_{y_2} \vdash \emptyset) ,$$

  where ψ' is an expression defined by Equation (31). Falsely assume that

  $$\psi' \rightarrow \exists x \in D'_x \exists y_1 \in D'_{y_1} \exists y_2 \in D'_{y_2} .\ x \notin \mathcal{X} \vee [y_1, y_2] \not\subseteq [\min(D'_{y_1}), b] .$$

  The part of x can be simplified such that

  $$\psi' \rightarrow \exists y_1 \in D'_{y_1} \exists y_2 \in D'_{y_2} .\ D'_x \setminus \mathcal{X} \neq \emptyset \vee [y_1, y_2] \not\subseteq [\min(D'_{y_1}), b] .$$

  But according to the construction of $\mathcal{X}$ we know that $D'_x \setminus \mathcal{X} = \emptyset$, so the expression is further simplified to

  $$\psi' \rightarrow \exists y_1 \in D'_{y_1} \exists y_2 \in D'_{y_2} .\ [y_1, y_2] \not\subseteq [\min(D'_{y_1}), b] ,$$

or

$$\psi' \rightarrow \exists y_1 \in D'_{y_1} \exists y_2 \in D'_{y_2}.\ y_1 \leq y_2 \wedge (y_1 < \min(D'_{y_1}) \vee b < y_2)\ .$$

Since y_1 is bounded only from above and y_2 is bounded from below then we can replace y_1 with $\min(D'_{y_1})$ and y_2 with $\max(D'_{y_2})$:

$$\psi' \rightarrow [\min(D'_{y_1}) \leq \max(D'_{y_2}) \wedge (\min(D'_{y_1}) < \min(D'_{y_1}) \vee b < \max(D'_{y_2}))]\ .$$

$\min(D'_{y_1}) < \min(D'_{y_1})$ is clearly false, and from the definition of ψ' we know that $b < \max(D'_{y_2})$ is false. This falsifies $\min(D'_{y_1}) < \min(D'_{y_1}) \vee b < \max(D'_{y_2})$ from the expression, hence the whole expression is also falsified. This means that our assumption was wrong, i.e., (9) is not violated.

To summarize, the rule is

$$\frac{y_1 \leq x \qquad [x, y_2] \not\subseteq [a, b] \qquad \psi}{a > x \geq \min(D'_{y_1}) \vee [y_1, y_2] \not\subseteq [\min(D'_{y_1}), b]}\ . \tag{35}$$

Rule R7: $\boldsymbol{c_1 \doteq (y \leq x + k_1)} \qquad \boldsymbol{c_2 = (x \leq y + k_2)}$

Isolating $x - y$ on both sides yields $c_{12}(x, y) = -k_1 \leq x - y \leq k_2$, which is false if $k_1 + k_2 < 0$. Since it is simply a conjunction of the input constraints, then (8) and (9) are satisfied trivially.

5 Experimental results

We compared three different settings: (1) HCSP with general constraints learning based on Combine (from hereon—HCSP), (2) HCSP using only clause-based learning with explanations, as described in Sec. 2 (from hereon—Explain)[4] (3) Mistral [8] latest version (1.550).

To evaluate the three alternatives we used a subset of benchmarks of the Fourth International CSP Solver competition [2]. Specifically out of over 7000 in the competition's satisfiability benchmark-set, we focused on the 2162 benchmarks that have at least one comparison operator from $\{<, \leq, \geq, >\}$ (the reason being that the rules in Table 1 refer to combinations of constraints based on these operators and constraints that are consequents of these rules).

All tools were compiled with gcc-4.8.1 for 32-bit Intel architecture, and were run on a 4 core Intel® Xeon® 2.5GHz with 3GiB RAM. We have set a hard limit of 1GiB memory usage and 1200 seconds of CPU-time. Out of memory and time-outs are called 'fails' in the discussion below.

[4] We emphasize that this is a far-improved engine in comparison to four years ago [12] owing to numerous optimizations that are beyond the scope of the current article, including an improved decision heuristic, better choice of explanations, and common-sub-expression elimination.

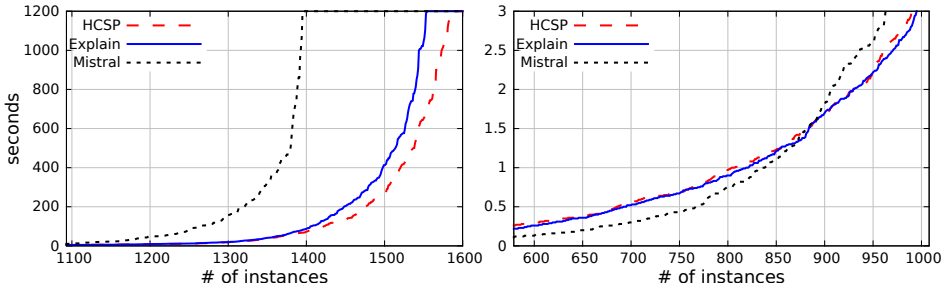


Fig. 2: Number of instances solved within the given time limit comparing HCSP, EXPLAIN, and MISTRAL (a) Shows the time in linear scale; (b) A zoom-in of the left figure showing the cross-over between MISTRAL and HCSP occurring after 1-2 seconds.

Fig. 2 compares the three engines. Number of fails in HCSP was 25% less than MISTRAL. Number of fails when using Combine was 4.9% less than EXPLAIN. In Fig.3(a) we see that the conflict analysis of HCSP is not beneficial for runs below 1.5 seconds, where MISTRAL ran faster.

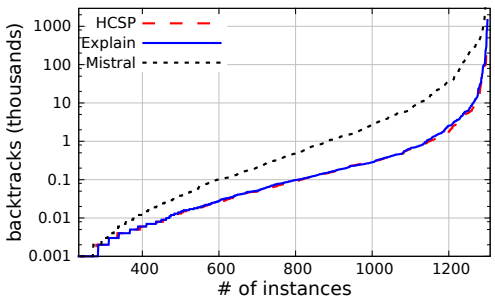


Fig. 3: Comparing the number of backtracks for successful runs (log-scale).

Fig. 3 compares the number of backtracks considering only non-failing runs in all solvers (log-scale). The average number of backtracks in HCSP is 2045, in EXPLAIN 4389, and in MISTRAL 49562. As noted in the introduction, this drastic difference in the average backtrack-count indicates that the cost of learning is compensated-for by a better search.

HCSP is written in C++, contains 23k lines of non-comment code, and its architecture enables the addition of new constraints and new rules without changing the core solver. It is free software [1] under the GPL license.

Conclusion and future work We have presented a new learning scheme based on inference of general constraints. We presented the development of various inference rules that are necessary for this scheme, but it is clear that there is still a lot of work in deriving such rules for additional popular pairs of constraints which are currently not supported and force HCSP into a fallback solution. In addition, currently learning general constraints is incompatible with producing machine-checkable proofs in case the formula is unsatisfiable, in contrast to our earlier explanation-based method [12].

References

1. HCSP Constraint Solver (version 1.0). `http://tx.technion.ac.il/~mveksler/hcsp/`.
2. Fourth international CSP solver competition. `http://cpai.ucc.ie/09/index.html`, 2009.
3. Bernhard Beckert, Reiner Hähnle, and Felip Manyá. The SAT problem of signed CNF formulas. pages 59–80, 2000.
4. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In Ramon López de Mántaras and Lorenza Saitta, editors, *ECAI*, pages 146–150. IOS Press, 2004.
5. Chiu Wo Choi, Warwick Harvey, J. H. M. Lee, and Peter J. Stuckey. Finite domain bounds consistency revisited. In Abdul Sattar and Byeong Ho Kang, editors, *Australian Conference on Artificial Intelligence*, volume 4304 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2006.
6. Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artif. Intell.*, 41(3):273–312, 1990.
7. Ian P. Gent, Ian Miguel, and Neil C. A. Moore. Lazy explanations for constraint propagators. In Manuel Carro and Ricardo Peña, editors, *PADL*, volume 5937 of *Lecture Notes in Computer Science*, pages 217–233. Springer, 2010.
8. Emmanuel Hebrard. Mistral, a constraints satisfiaction library. In *Third international CSP solver competition*, pages 31–40, 2008.
9. George Katsirelos and Fahiem Bacchus. Generalized nogoods in CSPs. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 390–396. AAAI Press / The MIT Press, 2005.
10. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. Design Automation Conference (DAC'01)*, 2001.
11. Ofer Shtrichman. Tuning SAT checkers for bounded model checking. 2000.
12. Michael Veksler and Ofer Strichman. A proof-producing csp solver. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.