

A Geometric Constraint over k -Dimensional Objects and Shapes Subject to Business Rules

Mats Carlsson¹, Nicolas Beldiceanu², and Julien Martin³

¹ SICS, P.O. Box 1263, SE-164 29 Kista, Sweden

`Mats.Carlsson@sics.se`

² École des Mines de Nantes, LINA UMR CNRS 6241, FR-44307 Nantes, France

`Nicolas.Beldiceanu@emn.fr`

³ INRIA Rocquencourt, BP 105, FR-78153 Le Chesnay Cedex, France

`Julien.Martin@inria.fr`

Abstract. This paper presents a global constraint that enforces rules written in a language based on arithmetic and first-order logic to hold among a set of objects. In a first step, the rules are rewritten to Quantifier-Free Presburger Arithmetic (QFPA) formulas. Secondly, such formulas are compiled to generators of k -dimensional forbidden sets. Such generators are a generalization of the indexicals of cc(FD). Finally, the forbidden sets generated by such indexicals are aggregated by a sweep-based algorithm and used for filtering.

The business rules allow to express a great variety of packing and placement constraints, while admitting effective filtering of the domain variables of the k -dimensional object, without the need to use spatial data structures.

1 Introduction

This paper extends a global constraint $geost(k, \mathcal{O}, \mathcal{S}, \mathcal{R})$ for handling the location in space of k -dimensional objects \mathcal{O} ($k \in \mathbb{N}^+$), each of which taking a shape among a set of shapes \mathcal{S} , subject to rules \mathcal{R} in a language based on arithmetic and first-order logic. This language can also be seen as a natural target constraint of the `Rules2CP` modeling language [1].

In order to model directly a lot of side constraints, which always show up in the context of real-life applications, many global constraints have traditionally been extended with extra options or arguments. This is why, in a closely related area, the *diffn* constraint [2] of CHIP provides, beside non-overlapping, a variety of other geometrical constraints (in fact more than 10 side constraints). Even if this makes sense when one wants to efficiently solve specific real-life applications, this proliferation of arguments and options has two major drawbacks:

- Having a lot of ad-hoc side constraints is too specific and can sometimes be quite frustrating since it does not allow to express a variant of an existing side constraint.
- Designing a filtering algorithm for each side constraint independently is not enough and managing the interaction of several side constraints becomes more and more challenging as the number and variety of side constraints increase.

The approach presented in this paper addresses these two issues in the following way:

- Firstly, having a rule language for expressing side constraints is obviously more flexible than having a large set of predefined side constraints.
- Secondly, as we will see later on, our filtering algorithms allow to directly take into account the interaction between all rules.

In $geost(k, \mathcal{O}, \mathcal{S}, \mathcal{R})$, each shape from \mathcal{S} is defined as a finite set of shifted boxes, where each shifted box is described by a box in a k -dimensional space at the given offset with the given sizes. More precisely a *shifted box* $s \in \mathcal{S}$ is an entity defined by its shape id $s.sid$, shift offset $s.t[d]$, $1 \leq d \leq k$, and sizes $s.l[d]$ (where $s.l[d] > 0$ and $1 \leq d \leq k$). All attributes of a shifted box are integer values. A *shape* is a collection of shifted boxes all sharing the same shape id.

Each object $o \in \mathcal{O}$ is an entity defined by its unique object id $o.oid$ (an integer), shape id $o.sid$ (an integer if the object has a fixed shape, or a domain variable for *polymorphic* objects, which have alternative shapes), and origin $o.x[d]$, $1 \leq d \leq k$ (integers, or domain variables that do not occur anywhere else in the constraint).¹ Objects and shifted boxes may also have additional, integer (but see also Section 6) attributes, such as weight, customer, or fragility, used by the rules.

Each rule in \mathcal{R} is a first-order logical formula over the attributes of objects and shifted boxes. From the point of view of domain filtering, the main contribution of this paper is that multi-dimensional forbidden sets can be automatically derived from such formulas and used by the sweep-based algorithm of *geost* [3].² This contrasts with the previous version of *geost*, where an ad-hoc algorithm computing the multi-dimensional forbidden sets had to be worked out for each side constraint. \mathcal{R} may also contain macros, providing abbreviations for expressions occurring in formulas or in other macros.

The rule language. The language that makes up the rules to be enforced by the *geost* constraint is based on first-order logic with arithmetic, as well as several features including macros, bounded quantifiers, folding and aggregation operators. We will show how all but a core fragment of the language can be eliminated by equivalence-preserving rewriting. The remaining fragment is a subset of Quantifier-Free Presburger Arithmetic (QFPA), which has a very simple semantics and, as we also will show, is amenable to efficient compilation.

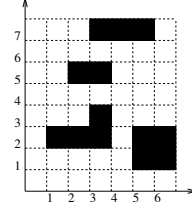
Constraint satisfaction problems using quantified formulas (QCSP) have for instance been studied by Benedetti et al. [4], mostly in the context of modeling games. QCSP does not provide disjunction but actively uses quantifiers in the evaluation, whereas we eliminate all quantifiers in the process of rewriting to QFPA.

Example 1. This running example will be used to illustrate the way we compile rules to code used by the sweep-based algorithm [3] for filtering the nonground attributes of each object. Suppose that we have five objects o_1, o_2, o_3, o_4 and o_5 such that:

¹ A *domain variable* v is a variable ranging over a finite set of integers denoted by $\text{dom}(v)$; \underline{v} and \overline{v} denote respectively the minimum and maximum possible values for v .

² The sweep-based algorithm performs recursive traversals of the placement space for each coordinate increasing as well as decreasing lexicographic order and skips unfeasible points that are located in a multi-dimensional forbidden set.

- o_1, o_2 and o_4 are rectangles fixed at $(1, 2)$, $(3, 3)$ and $(3, 7)$ of respective size 3×1 , 1×1 and 3×1 .
- The rectangle o_3 is fixed at $(2, 5)$ but not its shape variable s_3 , which can take values corresponding to size 1×2 or 2×1 . We will denote by ℓ_{31} resp. ℓ_{32} the length resp. height of o_3 .
- The coordinates of the non-fixed square o_5 of size 2×2 correspond to the two variables $x_{51} \in [1, 9]$ and $x_{52} \in [1, 6]$.
- o_2, o_4 and o_5 have the additional attribute *type* with value 1 whereas o_1 and o_3 have *type* with value 2.
- Two rules must be obeyed; see Fig. 1:
 - All objects should be mutually non-overlapping .
 - If the *type* attribute of two objects both equal 1, the two objects should not touch, not even their corners .
- The figure on the right shows one solution.



$$\begin{array}{l}
 \text{overlap}(D, o_i, s_i, o_j, s_j) \rightarrow \\
 \quad \forall d \in D : \text{end}(o_i, s_i, d) > \text{ori}(o_j, s_j, d) \wedge \text{end}(o_j, s_j, d) > \text{ori}(o_i, s_i, d) \\
 \text{meet}(D, o_i, s_i, o_j, s_j) \rightarrow \\
 \quad (\forall d \in D : \text{end}(o_i, s_i, d) \geq \text{ori}(o_j, s_j, d) \wedge \text{end}(o_j, s_j, d) \geq \text{ori}(o_i, s_i, d)) \wedge \\
 \quad (\exists d \in D : \text{end}(o_i, s_i, d) = \text{ori}(o_j, s_j, d) \vee \text{end}(o_j, s_j, d) = \text{ori}(o_i, s_i, d)) \\
 \text{all_not_overlap}(D, OIDs) \rightarrow \\
 \quad \forall o_i \in OIDs, \forall s_i \in o_i.\text{sid}, \forall o_j \in OIDs : \\
 \quad \quad o_i.\text{oid} < o_j.\text{oid} \Rightarrow (\forall s_j \in o_j.\text{sid} : \neg \text{overlap}(D, o_i, s_i, o_j, s_j)) \\
 \text{all_type1_not_meet}(D, OIDs) \rightarrow \\
 \quad \forall o_i \in OIDs, \forall s_i \in o_i.\text{sid}, \forall o_j \in OIDs : \\
 \quad \quad o_i.\text{oid} < o_j.\text{oid} \wedge o_i.\text{type} = 1 \wedge o_j.\text{type} = 1 \Rightarrow \\
 \quad \quad \forall s_j \in o_j.\text{sid} : \neg \text{meet}(D, o_i, s_i, o_j, s_j) \\
 \text{all_not_overlap_boxes}([1, 2], [1, 2, 3, 4, 5]) \\
 \text{all_type1_not_meet_boxes}([1, 2], [1, 2, 3, 4, 5])
 \end{array}$$

Fig. 1. Macros and rules of the running example. $\text{ori}(o, s, d)$ (resp. $\text{end}(o, s, d)$) stands for the origin (resp. end) in dimension d object o with shape s . \square

Declarative semantics. As usual, the semantics is given in terms of ground objects. The constraint $\text{geost}(k, \mathcal{O}, \mathcal{S}, \mathcal{R})$ holds if and only if the conjunction of the logical formulas in \mathcal{R} is true.

Implementation overview. Fig. 2 provides the overall architecture of the implementation. When the *geost* constraint is posted, the given business rules are translated, first into QFPA, then into generators of k -dimensional forbidden sets. Such generators, *k-indexicals*, are a generalization of the indexicals of cc(FD) [5]. Each time the constraint wakes up, the sweep-based algorithm [3] generates forbidden sets for a specific object o by invoking the relevant k -indexicals, then looks for points that are not contained in any forbidden set in order to prune the nonground attributes of o .

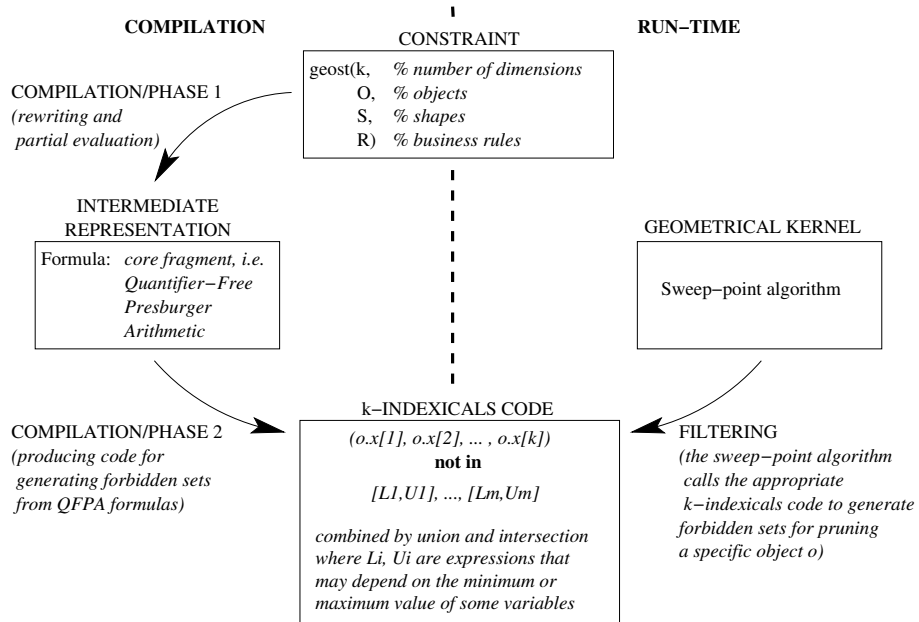


Fig. 2. Overall architecture of the implementation.

Paper outline. In Section 2, we present the rule language, its abstract syntax and its features. In Section 3, we present the QFPA core fragment of the language, its declarative semantics, and how the rule language is rewritten into QFPA. In Section 4, we describe (1) how a QFPA formula is compiled to generators of k -dimensional forbidden sets, and (2) how the forbidden sets generated by such generators are aggregated by a sweep-based algorithm and used for filtering. In Section 5, we provide experimental evidence for search space reduction due to the global treatment of side constraints. Before concluding, in Section 6, we mention a number of issues that we are currently working on. An expanded version of this paper is available as a technical report [6]. In particular, see [6, Section 5] for an extension of the filtering to accommodate polymorphic objects.

2 The Rule Language: Syntax and Features

A sentence is either a *macro* or a *fol*. A *macro* is simply a shorthand device: during a rewriting phase, whenever an expression matching the left-hand side of a macro is encountered, it is replaced by the corresponding right-hand side. A *fol* is a first-order logic formula that must hold for the constraint to be true, and is one of: a comparison between two arithmetic expressions, the constant `true` or `false`, a cardinality formula $\#(var, collection, integer, integer, fol)$, a quantified formula $\forall(var, collection, fol)$ or $\exists(var, collection, fol)$, or formulas combined with logical connectives: $\neg fol$, $fol \wedge fol$, $fol \vee fol$, $fol \Rightarrow fol$, or $fol \Leftrightarrow fol$.

An *expr* (arithmetic expression) is an integer, an *attref* (a reference to an attribute of an entity, where an *entity* is an object or a shifted box), a fold expression $@(var, collection, \circ, expr, expr)$ where $\circ \in \{+, \min, \max\}$, or an expression $expr \circ expr$ where $\circ \in \{+, -, \times, /, \min, \max\}$. Arithmetic expressions must be *linear*: in a product, at most one factor can be nonground; in a quotient, the divisor must be ground.

A *collection* is the shorthand $objects(S)$, denoting the collection of objects with object id in S , or the shorthand $sboxes(S)$ denoting the collection of shifted boxes with shape id in S , or a list of terms, where a *term* is a *variable*, an *integer*, an *identifier*, or a *compound term*. A *compound term* consists of a *functor* (an identifier) and one or more arguments (terms). A term is *ground* if it is free of variables.

Quantified formulas are meaningful if the quantified variable occurs in the quantified *fol*. They are treated by expansion to a disjunction resp. a conjunction of instances of that *fol* where each element of the *collection* is substituted for the quantified variable. In the context of our application, quantified variables typically vary over a collection of dimensions, objects, or shifted boxes.

A cardinality formula specifies a variable quantified over a list of terms, a lower and an upper bound, and a *fol* template mentioning the quantified variable. The formula is true if and only if the number of true instances of the *fol* template is within the given bounds. Cardinality formulas [7] are treated by expansion to \neg , \wedge and \vee connectives [8].

Arithmetic expressions and comparisons are over the rational numbers. The rationale for this is that business rules often involve fractions of measures like weight or volume. However, such fractions are converted to integers during rewriting.

Fold expressions allow to express e.g. the sum of some attribute over a set of objects. The operator specifies a variable quantified over a list of terms, a binary operator, an identity element, and a template mentioning the quantified variable. The identity element is needed for the empty list case.

3 QFPA Core Fragment

In this section, we show how a formula p in the rule language is rewritten by a series of equivalence-preserving transformations into a *qfpa*, i.e. a QFPA formula, which here either is of the form $\sum_i integer_i \cdot attref_i \geq integer$ or is a conjunction or a disjunction of *qfpas*.

QFPA is widely used in symbolic verification, and there has been much work on deciding whether a given QFPA formula is satisfiable [9]. Many methods based on integer programming techniques [10] rely on having the formula on disjunctive normal form. However, for constraint programming purposes, we are interested in necessary conditions that can be used for filtering domain variables, and we are not aware on any such work on QFPA. In [11], filtering algorithms for logical combinations of adhoc constraints³ are proposed, but it is not clear whether that approach can be extended to QFPA. For that, we would need to provide supports of *qfpas*.

³ Also known as constraints given in extension.

3.1 Rewriting into QFPA

We now show the details of rewriting the formula given as the *geost* parameter \mathcal{R} in the following eight steps into a *qfpa* $\hat{\mathcal{R}}$. We will later show how $\hat{\mathcal{R}}$ is translated to generators of forbidden sets.

Macro expansion and constant folding. The implication and equivalence connectives, bounded quantifiers, and cardinality and folding operators are eliminated. Ground integer expressions are replaced by their values. Object and shifted box collections are expanded.

Elimination of negation. Using DeMorgan's laws and negating relevant *relops*.

Normalization of arithmetic. Arithmetic relations are normalized to one of the forms $expr \geq 0$ or $expr > 0$.

Elimination of \times , $/$ and $-$. Any occurrence of these operators in arithmetic expressions is eliminated. At the same time, all operands are associated with a rational coefficient (c in the table). The elimination is made possible by the fact that in multiplication, at least one factor must be ground and is simply multiplied into the coefficient. Similarly, in division, the coefficient is simply divided by the divisor, which must be ground.

Moving $+$ inside min and max. Any expression with min or max occurring inside $+$ are rewritten by using the commutative and distributive laws (1) so that the $+$ is moved inside the other operator.

$$\begin{aligned} a + b &= b + a \\ a + \min(b, c) &= \min(a + b, a + c) \\ a + \max(b, c) &= \max(a + b, a + c) \end{aligned} \tag{1}$$

Elimination of min and max. Any min or max operators occurring in arithmetic relations are eliminated, replacing such relations by new relations combined by \wedge or \vee . After this step, an arithmetic expression is a linear combination of *attrefs* with rational coefficients, plus an optional constant.

Elimination of rational numbers. Any arithmetic relation r , which can now only be of the form $e > 0$ or $e \geq 0$, is normalized into the form $e'' \geq c''$ where e' and c' are intermediate expressions in:

- Let e' be the linear combination obtained by multiplying e by the least common multiplier of the denominators of the coefficients of e . Recall that those coefficients are rational numbers. Thus, the coefficients of e' are integers.
- Let c' be 1 if r is of the form $e > 0$, or 0 if r is of the form $e \geq 0$.
- If e' contains a constant term c , then $e'' = e' - c$ and $c'' = c' - c$. Otherwise, $e'' = e'$ and $c'' = c'$.

Simplification. Any entailed or disentailed arithmetic comparison is replaced by the appropriate logical constant (`true` or `false`). Any \wedge or \vee expression containing one of these constants is simplified using partial evaluation.

Example 2. Returning to our running example the resulting *qfpa* $\hat{\mathcal{R}}$ shown on the right is a conjunction of six subformulas corresponding respectively to:

- From the business rule `all_not_overlap_sboxes`, conditions to prevent o_5 from overlapping o_1, o_2, o_3 and o_4 .
- From the business rule `all_type1_not_meet_sboxes`, conditions to prevent o_5 from meeting o_2 and o_4 .

□

$$\left(\begin{array}{c} \vee \left(\begin{array}{c} x_{51} \geq 4 \\ x_{52} \geq 3 \end{array} \right) \\ \vee \left(\begin{array}{c} x_{51} \geq 4 \\ -1 \cdot x_{51} \geq -1 \\ x_{52} \geq 4 \\ -1 \cdot x_{52} \geq -1 \end{array} \right) \\ \vee \left(\begin{array}{c} -1 \cdot \ell_{31} + x_{51} \geq 2 \\ -1 \cdot \ell_{32} + x_{52} \geq 5 \\ -1 \cdot x_{52} \geq -3 \end{array} \right) \\ \vee \left(\begin{array}{c} x_{51} \geq 6 \\ -1 \cdot x_{51} \geq -1 \\ -1 \cdot x_{52} \geq -5 \end{array} \right) \\ \wedge \left(\begin{array}{c} x_{51} \geq 5 \\ x_{52} \geq 5 \\ \vee \left(\begin{array}{c} -1 \cdot x_{51} \geq -3 \\ x_{51} \geq 5 \end{array} \right) \\ \wedge \left(\begin{array}{c} x_{51} \geq 2 \\ -1 \cdot x_{52} \geq -3 \\ x_{52} \geq 5 \end{array} \right) \\ x_{52} \geq 2 \end{array} \right) \\ \vee \left(\begin{array}{c} x_{51} \geq 7 \\ -1 \cdot x_{52} \geq -4 \\ \vee \left(\begin{array}{c} -1 \cdot x_{51} \geq -5 \\ x_{51} \geq 7 \end{array} \right) \\ \wedge \left(\begin{array}{c} x_{51} \geq 2 \\ x_{52} \geq 6 \\ -1 \cdot x_{52} \geq -4 \end{array} \right) \end{array} \right) \end{array} \right)$$

4 Compiling to an Efficient Run-Time Representation

It is straightforward to obtain necessary conditions for *qfpas* as well as pruning rules operating on one variable at a time. Based on such conditions and pruning rules, we will show how to construct generators of k -dimensional forbidden sets. We call such generators *k-indexicals*, for they are generalization of the indexicals of cc(FD) [5]. Finally, we show how the forbidden sets generated by such indexicals are aggregated by the sweep-based algorithm [3] and used for filtering.

Indexicals were first introduced for the language cc(FD) [5] and later used in the context of CLP(FD) [12, 13], AKL [14], finite set constraints [15] and adhoc constraints [16]. They have proven a powerful and efficient way of implementing constraint propagation. A key feature of an indexical is that it is a function of the current domains of the variables on which it depends. Thus, indexicals also capture the propagation from variables to variables that occurs as variables are pruned. In the cited implementations,

an indexical is a procedure that computes the feasible set of values for a variable. We generalize this notion to generating a forbidden set of k -dimensional points, for an object, and so k -indexicals captures the propagation from objects to objects that occurs as object attributes are pruned.

4.1 Necessary Conditions

For a formula R denoting a linear combination of variables, let $MAX(R)$ denote the expression that replaces every *attref* x in R by \bar{x} if x occurs with a positive coefficient, and by \underline{x} otherwise. Thus, $MAX(R)$ is a formula that computes an upper bound of R wrt. the current domains.

We will ignore the degenerate cases where $\hat{\mathcal{R}}$ is `true` resp. `false`, in which case *geost* merely succeeds resp. fails. For the normal *qfpa* cases, we obtain the necessary conditions shown in Table 1.

<i>qfpa t</i>	necessary condition $N(t)$
$\sum_i c_i \cdot x_i \geq r$	$MAX(\sum_i c_i \cdot x_i) \geq r$
$p \vee q$	$N(p) \vee N(q)$
$p \wedge q$	$N(p) \wedge N(q)$

Table 1. Necessary condition $N(t)$ for *qfpa t*

4.2 Pruning Rules

For the base case $\sum_i c_i \cdot x_i \geq r$, we have the well-known pruning rules (2), which provide sharp bounds; see e.g. [17] for details.

$$\forall j \begin{cases} x_j \geq \lceil \frac{r - MAX(\sum_{i \neq j} c_i \cdot x_i)}{c_j} \rceil, & \text{if } c_j > 0 \\ x_j \leq \lfloor \frac{-r + MAX(\sum_{i \neq j} c_i \cdot x_i)}{-c_j} \rfloor, & \text{otherwise} \end{cases} \quad (2)$$

Consider now a disjunction $p \vee q$ of two base cases and a variable x_j occurring in at least one disjunct.

- If x_j occurs in p but not in q , rule (2) is only valid for p if the necessary condition for q does not hold.
- Similarly if x_j occurs in q but not in p .
- If x_j occurs in both p and q , we can use rule (2) for both p and q and conclude that x_j must be in the union of the two feasible intervals.

Finally, consider a conjunction $p \wedge q$, i.e. both p and q must hold. If x_j occurs in both p and q , we can use rule (2) for both p and q and conclude that x_j must be in the intersection of the two feasible intervals.

Example 3. Returning to our running example, consider the fragment $x_{51} \geq 4 \vee x_{52} \geq 3$ of the *qffa*, which comes from a rule preventing o_5 from overlapping o_1 . Suppose that we want to prune x_{52} . Then we can combine the necessary condition for $x_{51} \geq 4$ with rule (2) for $x_{52} \geq 3$ into the conditional pruning rule:

$$\max(x_{51}) < 4 \Rightarrow x_{52} \geq 3$$

However, as we will show in the next section, instead of using such conditional pruning rules, we unify necessary conditions and pruning rules into multi-dimensional forbidden sets and aggregate them per object. For the above fragment, the two-dimensional forbidden set for o_5 is $([1, 3], [1, 2])$, denoting the fact that (x_{51}, x_{52}) should be distinct from all the pairs $(1, 1), (1, 2), (2, 1), (2, 2), (3, 1), (3, 2)$. \square

4.3 *k*-Indexicals

Recall that the set of rules given in \mathcal{R} has been rewritten into a *qffa* $\hat{\mathcal{R}}$. Consider this formula, or some subformula $\hat{\mathcal{R}}_i$ of it if $\hat{\mathcal{R}}$ is a conjunction (see Section 4.4). The idea is to compile this subformula, for each object o mentioned by it, into a *k*-indexical for $\hat{\mathcal{R}}_i$ and o . The forbidden sets that it generates can then be aggregated and used by the sweep-point kernel [3] to prune the nonground attributes of o . Let us introduce some notation to make this idea clear.

Definition 1. A forbidden set for a *qffa* r and object o is a set⁴ of *k*-dimensional points such that, if o is placed at any of these points, r is disentailed.

Such a forbidden set can also be seen as the multi-dimensional generalization of a set of inconsistent assignments [18].

Definition 2. A *k*-indexical for a *qffa* r and an object o is a procedure that functions as a generator of forbidden sets for r and o . It is of the form $o.x \notin \text{ibody}$ where *ibody* is defined in Fig. 3. The *k*-indexical depends on object o' if *ibody* mentions o' .

k-indexicals are described by the inductive definition shown in Fig. 3. They are built up from generators of *k*-dimensional half-planes, combined by union and intersection operations.

4.4 Compilation

The *qffa* $\hat{\mathcal{R}}$, normally⁵ a conjunction $\hat{r}_1 \wedge \dots \wedge \hat{r}_n$, is compiled to *k*-indexicals by the following steps:

⁴ A forbidden set is not explicitly represented as a set of points, but rather by a set of boxes, as is the case in the earlier implementation [3].

⁵ Since it comes from the conjunction of business rules stated in the last argument of *geost*.

$k\text{-indexical}$	$::=$	$object.x \notin ibody$	
$ibody$	$::=$	$ibody \cap ibody$	
		$ibody \cup ibody$	
		$\{p \in \mathbb{Z}^k \mid p[d] < \lceil \frac{integer - \sum ubterm}{usi} \rceil\}$	
		$\{p \in \mathbb{Z}^k \mid p[d] > \lfloor \frac{integer + \sum ubterm}{usi} \rfloor\}$	
		if $\sum ubterm < r$ then \mathbb{Z}^k else \emptyset	
$ubterm$	$::=$	$usi \cdot \overline{attref}$	
		$-usi \cdot \overline{attref}$	
		$integer$	
d	$::=$	$integer$	{ denoting a dimension }
usi	$::=$	$integer$	{ > 0 }

Fig. 3. k -indexicals

1. Partition the conjuncts of $\hat{\mathcal{R}}$ into equivalence classes $\hat{\mathcal{R}}_1, \dots, \hat{\mathcal{R}}_m$ such that for all $1 \leq i < j \leq n$, \hat{r}_i and \hat{r}_j are in the same equivalence class if and only if they mention⁶ the same set of objects of \mathcal{O} .
2. For each equivalence class $\hat{\mathcal{R}}_i$ and object $o \in \mathcal{O}$ mentioned by $\hat{\mathcal{R}}_i$, map $\hat{\mathcal{R}}_i$ (as a conjunction) into a k -indexical for o , of the form $o.x \notin F_o(\hat{\mathcal{R}}_i)$, according to Table 2.

The mapping closely follows the pruning rules (2), except now we want to obtain a forbidden set instead of a feasible interval. Rows 1-2 of Table 2 are analogous to the recursive computation of inconsistent assignments in [18, Table 1]. Row 5 corresponds to the case where r does not mention o , in which case all points are forbidden for o if r is disentailed, and no points are forbidden for o otherwise.

The rationale for aggregating the conjuncts into equivalence classes, as opposed to mapping one conjunct at a time, is the opportunity to increase the granularity of the indexicals and to merge subformulas coming from different business rules. This opens the scope for future work on global simplification of formulas, and increases the amount of subexpressions that can be shared within a k -indexical.

It is well known that indexicals can be efficiently compiled and executed by a virtual machine [12, 13]. In our context, we predict that there will be a large amount of common subterms in the k -indexicals, and so common subexpression elimination will be quite important. Therefore, a register-based virtual machine would seem an appropriate choice.

Example 4. Returning to our running example, we obtained a *qfpa* which was a conjunction of six subformulas. They are partitioned into two equivalence classes: one for

⁶ A formula *mentions* an object o if it refers to a nonground attribute of o .

r	F_o(r)	condition
$p \vee q$	$F_o(p) \cap F_o(q)$	
$p \wedge q$	$F_o(p) \cup F_o(q)$	
$\sum_i c_i \cdot x_i \geq r$	$\{p \in \mathbb{Z}^k \mid p[d] < \lceil \frac{r - \text{MAX}(\sum_{i \neq j} c_i \cdot x_i)}{c_j} \rceil\}$	$x_j = o.x[d], c_j > 0$
$\sum_i c_i \cdot x_i \geq r$	$\{p \in \mathbb{Z}^k \mid p[d] > \lfloor \frac{-r + \text{MAX}(\sum_{i \neq j} c_i \cdot x_i)}{-c_j} \rfloor\}$	$x_j = o.x[d], c_j < 0$
$\sum_i c_i \cdot x_i \geq r$	if $\text{MAX}(\sum_i c_i \cdot x_i) < r$ then \mathbb{Z}^k else \emptyset	$o.x[d] \notin \{x_i\}$

Table 2. Mapping a *qfpa* r to a generator of forbidden sets, $F_o(r)$, for the object o . We assume here that o is not polymorphic.

the single conjunct that mentions both o_3 and o_5 , mapped to k -indexicals (3) and (4) below; and one for the five conjuncts that only mention o_5 (because o_1, o_2 and o_4 are ground), mapped to k -indexical (5) below. The three k -indexicals reflect the following business rules:

1. o_3 must not take a shape that will cause it to overlap o_5 . Note that this k -indexical propagates from o_5 to the shape id of o_3 . Pruning of shape ids of polymorphic objects is discussed in [6, Section 5]. Initially, no forbidden boxes are generated.

$$s_3 \notin \bigcap \left(\begin{array}{l} \{i \in \text{dom}(s_3) \mid s_3 = i \Rightarrow \ell_{31} > \overline{x_{51}} - 2\} \\ \{i \in \text{dom}(s_3) \mid s_3 = i \Rightarrow \ell_{32} > \overline{x_{52}} - 5\} \\ \text{if } \underline{x_{52}} > 3 \text{ then } \mathbb{Z} \text{ else } \emptyset \end{array} \right) \quad (3)$$

2. o_5 must not overlap o_3 . Note that this k -indexical propagates from o_3 to o_5 .

$$o_5.x \notin ([1, (\underline{\ell}_{31} + 1)], [4, (\underline{\ell}_{32} + 4)]) \quad (4)$$

3. o_5 must not overlap o_1, o_2 nor o_4 , nor meet o_2 nor o_4 .

$$o_5.x \notin \bigcup \left(\begin{array}{l} ([1, 3], [1, 2]) \\ ([2, 3], [2, 3]) \\ ([2, 5], [6, 6]) \\ ([1, 4], [1, 4]) \\ ([4, 4], [1, 6]) \\ ([1, 1], [1, 6]) \\ ([1, 9], [4, 4]) \\ ([1, 9], [1, 1]) \\ ([1, 6], [5, 6]) \\ ([1, 9], [5, 5]) \\ ([6, 6], [1, 6]) \\ ([1, 1], [1, 6]) \end{array} \right) \quad (5)$$

□

4.5 Filtering Algorithm

We now give a sketch of a filtering algorithm for $geost(k, \mathcal{O}, \mathcal{S}, \mathcal{R})$. Let $I(o)$ denote the set of k -indexicals for object $o \in \mathcal{O}$ wrt. the given rules \mathcal{R} , let $eval(i)$ denote the evaluation of k -indexical i wrt. the current domains, let $sweep(o, F)$ denote the application of the sweep-based algorithm to the object o wrt. the forbidden set F , which prunes the minimum and maximum values of the origin coordinates of o . Our proposed Algorithm 1 is a straightforward propagation loop.

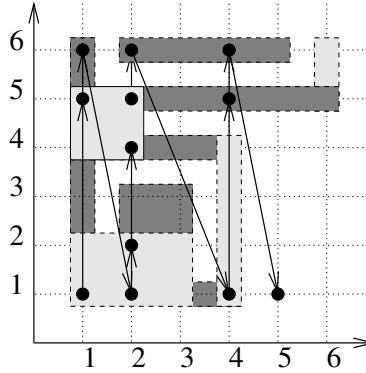
```

PROCEDURE Filter( $\mathcal{O}, I$ )
1:  $Q \leftarrow \mathcal{O}$ 
2: while  $Q \neq \emptyset$  do
3:    $o \leftarrow$  some element from  $Q$ 
4:    $Q \leftarrow Q \setminus \{o\}$ 
5:    $F \leftarrow \bigcup \{eval(i) \mid i \in I(o)\}$ 
6:   if  $\neg sweep(o, F)$  then
7:     return fail
8:   else if a coordinate of  $o$  was pruned then
9:      $Q \leftarrow Q \cup \{o' \mid I(o') \text{ depends on } o\}$ 
10:  end if
11: end while
12: if all objects in  $\mathcal{O}$  are ground then
13:  return succeed
14: else
15:  return suspend
16: end if

```

Algorithm 1: Sketch of a filtering algorithm for $geost(k, \mathcal{O}, \mathcal{S}, \mathcal{R})$

Example 5. Returning to our running example, suppose now that the sweep-point kernel wants to adjust the lower bound of x_{51} . The figure on the right traces the steps performed by the algorithm when it walks from a lexicographically smallest position to the first feasible position of o_5 . The result is that the lower bound of x_{51} is adjusted to 5. \square



5 Experimental Results

The $geost$ constraint, including the rewriting, compilation, and sweep-based algorithms, has been implemented in SICStus Prolog 4 [19] using its global constraint programming

API. A direct performance comparison of this proof-of-concept implementation with the earlier implementation [3], coded in C, is not meaningful. Therefore, we focus on showing the potential for search space reduction and stronger filtering due to the global treatment of side constraints.

We studied the placement problem given in [6, Appendix C], provided by Peugeot Citroën, which involves a $1203 \times 235 \times 239$ container and 9 objects with an extra *weight* attribute, subject to the rules:

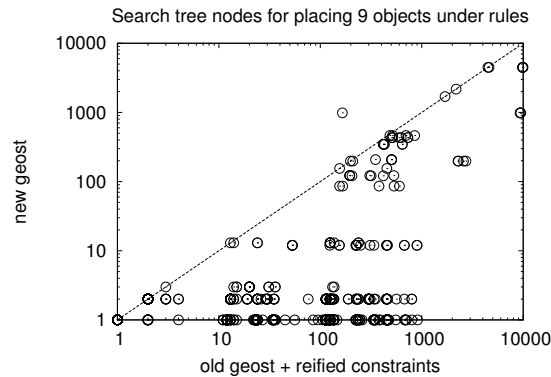
- (a) Each object is placed inside the container.
- (b) Each object is either on the floor or resting on some other object.
- (c) The objects do not pairwise overlap.
- (d) A heavier object cannot be piled on top of a lighter one.
- (e) For any two objects in a pile, the overhang can be at most 10 units.

In [6] we provide encodings of these rules as well as other rules encoding a packing-unpacking problem with visibility constraints⁷ and the time dimension.

We generated 600 problem instances by randomly permuting the objects. The search was performed by labeling the coordinates, grouped by object, in the permuted order, under a time limit of one CPU minute. For each instance, we posted the constraint, and measured the number of search space nodes visited during search for the first solution. Each instance was run twice:

1. with the new *geost* constraint, and
2. without it, but expressing constraint (c) with the earlier implementation [3] and constraints (a, b, d, e) with logical combinations of arithmetic constraints.

The scatter plot shown on the right summarizes the results. Each dot represents an instance, its X (resp. Y) coordinate corresponding to the old (resp. new) implementation. The search effort was decreased by 100 times or more in 26% of the cases and by 10 times or more in another 33% of the cases.



6 Discussion

Generality. Our restriction that object attributes (except shape id and origin) must be ground is somewhat artificial, and we plan to lift it. The rewritten QFPA formulas would

⁷ See the *visible* constraint in <http://www.emn.fr/x-info/sdemasse/gccat/>.

simply have more variables per object, and the sweep-based algorithm would deal not with a k - or $k + 1$ -dimensional *placement* space, but with an m -dimensional *solution* space, where m is the number of possibly nonground attributes per object. In particular, in order to deal with objects whose length in some dimension is a domain variable that occurs in some other constraint, the length and possibly the end-point would have to be expressed as nonground object attributes. Similarly, to treat the time dimension, we would add three nonground object attributes *start*, *duration*, and *completion*, as in [3], to be included in the solution space.

Theoretical properties. It has been shown [1, Proposition 1-2] that the PKML/Rules2CP rewriting system is confluent and Noetherian (i.e., terminating). Since our rule language is essentially a subset of Rules2CP, the results apply to *geost* rules as well. A size bound on programs generated from Rules2CP is also known [1, Proposition 3] and applies to *geost* provided that min, max and cardinality is not used in the rules, since these operators can cause an exponential (for min and max) resp. quadratic (for cardinality) [8] blow-up. Consequently, one can certainly construct pathological cases where the rewrite phases and/or runtime representation require huge amounts of memory. Even if, at this time, this has not really been a problem for the instances and rules we have experimented with⁸, one way to manage the complexity of the rewrite phases is to apply simplifying rewrites, e.g. Phase 8, as eagerly as possible. Another way could be to memoize patterns that have already been rewritten. Finally, common subexpression elimination will mitigate this problem.

7 Conclusion

We have presented a global constraint that enforces rules written in a language based on arithmetic and first-order logic to hold among a set of objects. By rewriting the rules to QFPA formulas, we have shown how to compile them to k -indexicals and how the forbidden sets generated by such indexicals can be aggregated by a sweep-based algorithm and used for filtering. Initial experiments support the feasibility of the approach. The approach combines an expressive logic-based rule modeling language for stating business rules with a generic geometrical algorithm for effective filtering.

Finally, QFPA is a language in which also many other problems, unrelated to packing and placement, can be stated. In this paper, we have begun to explore a way to compile and run it efficiently.

Acknowledgements

This research was conducted under European Union Sixth Framework Programme Contract FP6-034691 “Net-WMS”. The second author was also partly supported by ANR (CANAR/06-BLAN-0383-02).

⁸ They involved at most 100 objects.

References

1. F. Fages and J. Martin. From rules to constraint programs with the `Rules2CP` modelling language. Research Report RR-6495, INRIA Rocquencourt, 2008.
2. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97–123, 1994.
3. N. Beldiceanu, M. Carlsson, E. Poder, R. Sadek, and C. Truchet. A generic geometrical constraint kernel in space and time for handling polymorphic k -dimensional objects. In C. Bessière, editor, *Proc. CP'2007*, volume 4741 of *LNCS*, pages 180–194. Springer, 2007.
4. M. Benedetti, A. Lallouet, and J. Vautard. QCSP made practical by virtue of restricted quantification. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 38–43, 2007.
5. P. Van Hentenryck, V. Saraswat, and Y. Deville. Constraint processing in cc(FD). unpublished manuscript, Computer Science Department, Brown University, 1991.
6. N. Beldiceanu, M. Carlsson, and J. Martin. A geometric constraint over k -dimensional objects and shapes subject to business rules. SICS Technical Report T2008:04, Swedish Institute of Computer Science, 2008.
7. P. Van Hentenryck and Y. Deville. The *cardinality* operator: a new logical connective in constraint logic programming. In *Int. Conf. on Logic Programming (ICLP'91)*. MIT Press, 1991.
8. N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–25, 2006.
9. V. Ganesh, S. Berezin, and D.L. Hill. Deciding presburger arithmetic by model checking and comparisons with other methods. In *Proc. FMCAD'02*, volume 2517 of *LNCS*, pages 171–186. Springer, 2002.
10. W. Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Supercomputing*, pages 4–13, 1991.
11. O. Lhomme. Arc-consistency filtering algorithms for logical combinations of constraints. In Jean-Charles Régin and Michel Rueher, editors, *Proc. CP-AI-OR 2004*, volume 3011 of *LNCS*, pages 209–224. Springer, 2004.
12. P. Codognet and D. Diaz. Compiling constraints in clp(FD). *Journal of Logic Programming*, 27(3):185–226, 1996.
13. M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In H. Glaser, P. Hartel, and H. Kuchen, editors, *Programming Languages: Implementations, Logics, and Programming*, volume 1292 of *LNCS*, pages 191–206. Springer, 1997.
14. B. Carlson. *Compiling and Executing Finite Domain Constraints*. PhD thesis, Uppsala University, 1995.
15. G. Tack, C. Schulte, and G. Smolka. Generating propagators for finite set constraints. In F. Benhamou, editor, *Proc. CP'2006*, volume 4204 of *LNCS*, pages 575–589. Springer, 2006.
16. K.C.K. Cheng, J.H.M. Lee, and P.J. Stuckey. Box constraint collections for adhoc constraints. In F. Rossi, editor, *Proc. CP'2003*, volume 2833 of *LNCS*, pages 214–228. Springer, 2003.
17. W. Harvey and P. J. Stuckey. Constraint representation for propagation. In M. Maher and J.-F. Puget, editors, *Proc. CP'98*, volume 1520 of *LNCS*, pages 235–249. Springer, 1998.
18. F. Bacchus and Toby Walsh. Propagating logical combinations of constraints. In *IJCAI-05, Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 35–40, 2005.
19. M. Carlsson et al. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, release 4 edition, 2007. ISBN 91-630-3648-7.