

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

M. Carlsson, SICS

N. Beldiceanu, EMN LINA UMR CNRS 6241

J. Martin, INRIA Rocquencourt



September 18, 2008

Outline

Introduction

- Motivation & Context
- The *geost* Constraint

The Rule Language

- Features
- Rewriting to Core Fragment

Filtering

- Compilation & Propagator

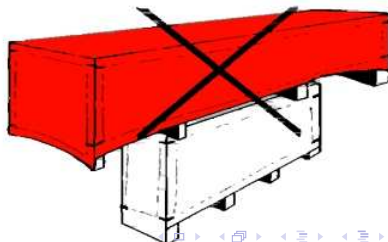
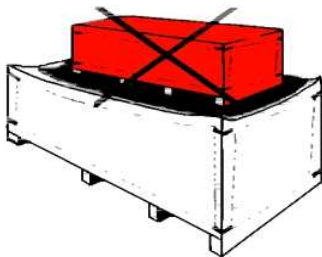
Evaluation and Concluding Remarks

- Evaluation
- Concluding Remarks

Motivation

Packing Rules: Example

“La caisse, ou le groupe de caisse, qui est placée en dessous d’une autre caisse et dont la dimension au sol n’est pas identique à la dimension au sol de la caisse à gerber à 10cm près (écart paramétrable) ne doivent pas être gerbées ensemble.”



Key Ideas in NET-WMS

Describing the Problem

Use a limited number of important **core constraints**.

Express **side constraints** with rules.

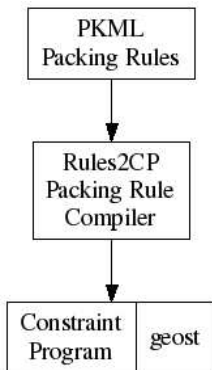
Performing Propagation

Rules are compiled to code that computes infeasible combinations of values (k -indexicals).

A **sweep algorithm** aggregates the information in order to prune the variables.

Context

Packing Rules \implies Constraint Program Armed with *geost*



The *geost* Constraint

$$geost(k, \mathcal{O}, \mathcal{S}, \mathcal{R})$$

- k Number of dimensions.
- \mathcal{O} Set of objects o with unique **object id** $o.oid$ (an integer), **shape id** $o.sid$, **origin** $o.x[d]$, $1 \leq d \leq k$, **optionally more** integer attributes.
- \mathcal{S} Set of shifted boxes s with **shape id** $s.sid$, **shift offset** $s.t[d]$, $1 \leq d \leq k$, **size** $s.l[d]$ ($s.l[d] > 0$, $1 \leq d \leq k$), **optionally more** attributes (all integers).
- \mathcal{R} Set of rules involving objects of \mathcal{O} .

Semantics Ground instance is true iff all given rules are true.

The Rule Parameter

- ▶ Previously, \mathcal{R} was a set of predefined “external constraints”.
- ▶ Now, \mathcal{R} is a set of statements in a first order logic based rule language (arithmetic constraints replace predicates).

2-Layered Design

full language Many features, rewritten into core fragment.

core fragment Subset of Quantifier-Free Presburger Arithmetic (QFPA). Small, clear semantics, amenable to compilation.

Paper's Running Example

NB. Object o_3 is *polymorphic*

Objects (4 attributes)

Id	Shape	Origin	Type
o_1	3×1	(1, 2)	2
o_2	1×1	(3, 3)	1
o_3	1×2 or 2×1	(2, 5)	2
o_4	3×1	(3, 7)	1
o_5	2×2	(1..9, 1..6)	1

Rules

- ▶ All objects should be mutually non-overlapping.
- ▶ If two objects are both of type 1, they should not touch, not even their corners.

Paper's Running Example

geost encoding: objects, shapes, macros

```
S3 in 3..4, X51 in 1..9, X52 in 1..6,
geost([object(1, 1,[ 1, 2],[type-2]),
      object(2, 2,[ 3, 3],[type-1]),
      object(3,S3,[ 2, 5],[type-2]),
      object(4, 1,[ 3, 7],[type-1]),
      object(5, 5,[X51,X52],[type-1])),
[sbox(1,[0,0],[3,1]),
 sbox(2,[0,0],[1,1]),
 sbox(3,[0,0],[1,2]),
 sbox(4,[0,0],[2,1]),
 sbox(5,[0,0],[2,2])],
[(origin(O1,S1,D) ---> O1^x(D)+S1^t(D)),
 (end(O1,S1,D) ---> O1^x(D)+S1^t(D)+S1^l(D)),
 MoreRules...]).
```

Paper's Running Example

geost encoding: macros and rule for non-overlapping

```
(overlap_sboxes(Dims,O1,S1,O2,S2) --->
  forall(D,Dims,
    end(O1,S1,D) #> origin(O2,S2,D) #/\
    end(O2,S2,D) #> origin(O1,S1,D))),
(all_not_overlap_sboxes(Dims,OIDs) --->
  forall(O1,objects(OIDs),
    forall(S1,sboxes([O1^sid])),
      forall(O2,objects(OIDs),
        O1^oid #< O2^oid #=>
          forall(S2,sboxes([O2^sid])),
            #\ overlap_sboxes(Dims,O1,S1,O2,S2))))),
all_not_overlap_sboxes([1,2],[1,2,3,4,5]),
```

Paper's Running Example

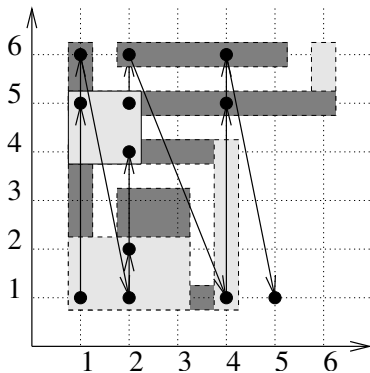
geost encoding: macros and rule for non-touching

```
(meet_sboxes(Dims,O1,S1,O2,S2) --->
  forall(D,Dims,
    end(O1,S1,D) #>= origin(O2,S2,D) #/\
    end(O2,S2,D) #>= origin(O1,S1,D)) #/\
  exists(D,Dims,
    end(O1,S1,D) #= origin(O2,S2,D) #\
    end(O2,S2,D) #= origin(O1,S1,D))),
(all_type1_not_meet_sboxes(Dims,OIDs) --->
  forall(O1,objects(OIDs),
    forall(S1,sboxes([O1^sid]),
      forall(O2,objects(OIDs),
        O1^oid #< O2^oid #/\ O1^type#=1 #/\ O2^type#=1 #=>
          forall(S2,sboxes([O2^sid]),
            #\ meet_sboxes(Dims,O1,S1,O2,S2)))))),
all_type1_not_meet_sboxes([1,2],[1,2,3,4,5]),
```

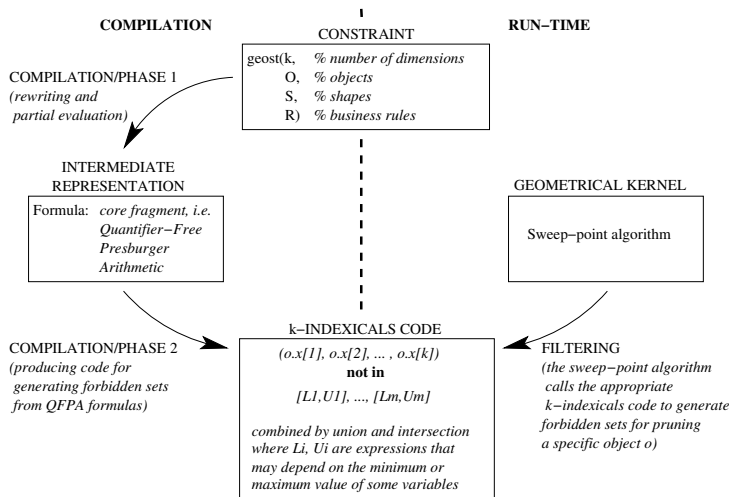
Paper's Running Example

sweep: Seeking a point outside of all forbidden regions

The lower bound of X_{51} is adjusted to 5 according to *all* rules (non-overlapping, non-touching).



Overall Architecture



Language Features

- ▶ A macro $head \implies body$ provides an abbreviation for $body$ occurring anywhere, Any instance of a macro $head$ is replaced by the corresponding instance of $body$.
- ▶ $\implies, \iff, \exists, \forall, \#$, expanding to \neg, \wedge, \vee .
- ▶ Cardinality operator $\#$, folding operator $@$.

$$\text{origin}(o, s, d) \implies o.x[d] + s.t[d]$$

formula	equivalent
$\text{origin}(o_1, s_2, 3)$	$o_1.x[3] + s_2.t[3]$
$\exists(x, [1, 2, 3], p(x))$	$p(1) \vee p(2) \vee p(3)$
$\forall(x, [1, 2, 3], p(x))$	$p(1) \wedge p(2) \wedge p(3)$
$\#(x, [o_1, o_2], 1, 1, x.u > 5)$	$(o_1.u > 5 \wedge o_2.u \leq 5) \vee (o_1.u \leq 5 \wedge o_2.u > 5)$
$@(x, [o_1, o_2, o_3], +, 0, x.u)$	$o_1.u + o_2.u + o_3.u$

Core Fragment

Subset of Quantifier-Free Presburger Arithmetic (QFPA).

$$\begin{array}{lll} qfpa & ::= & qfpa \wedge qfpa & \{\text{conjunction}\} \\ & | & qfpa \vee qfpa & \{\text{disjunction}\} \\ & | & \sum_i integer_i \cdot attref_i \geq integer & \{\text{base case}\} \\ \\ attref & ::= & entity.attref & \{\text{nonground reference}\} \end{array}$$

An *attref* corresponds to a nonground attribute of an object (i.e. an origin) or an attribute of a shifted box of a polymorphic object (i.e. a size).

Rewriting into the Core Fragment

1. Eliminate $\forall, \exists, \#, \Rightarrow, \Leftrightarrow, @$, macros, ground references, `objects(list)`, `sboxes(list)`.
2. Eliminate \neg .
3. Eliminate $<, \leq, =, \neq$.
4. Eliminate $\times, /, -$.
5. Move up any min, max occurring inside $+$.
6. Eliminate min, max, replacing with \wedge, \vee .
7. Eliminate rational numbers and $>$.
8. Simplify away true/false subformulas.

Aggregation

For the sweep algorithm

Forbidden set for *qfpa* r and object o

A set of k -dimensional points such that if o is placed at any of these points, r is disentailed.

k -indexical for *qfpa* r and object o

- ▶ A generator of forbidden sets for *qfpa* r and object o .
- ▶ Evaluated wrt. the current domains of \mathcal{O} .
- ▶ Captures propagation to o from objects on which o depends.
- ▶ Can be compiled.

Mapping $qfpa\ r$ to k -Indexicals for Object o

Mapping o, r to $o.x \notin F_o(r)$

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

In the paper, we explain how to generalize this to the polymorphic case, allowing to solve for shape id $o.sid$.

Efficiency issues

Prototype implementation in Prolog

- ▶ k -Indexicals can be compiled to virtual machine code procedures + dependencies.
- ▶ A register-based virtual machine supports common subexpression elimination.
- ▶ \cap and \cup give rise to conditional jumps.
- ▶ If r is ground and $F_o(r)$ is disjoint from $\text{dom}(o.x)$, the k -indexical $o.x \notin F_o(r)$ can be disabled (entailment).

Propagator

notation	meaning
$I(o)$	set of k -indexicals for object $o \in \mathcal{O}$
$\text{eval}(i)$	evaluation of k -indexical i
$\text{sweep}(o, F)$	application of the sweep algorithm on o, F

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:    $F \leftarrow \bigcup \{\text{eval}(i) \mid i \in I(o)\}$ 
6:   if  $\neg \text{sweep}(o, F)$  then
7:     return fail
8:   else if  $o$  was pruned then
9:      $Q \leftarrow Q \cup \{o' \mid I(o') \text{ depends on } o\}$ 
10:  end if
11: end while
12: return suspend
```

Evaluation of Effectiveness

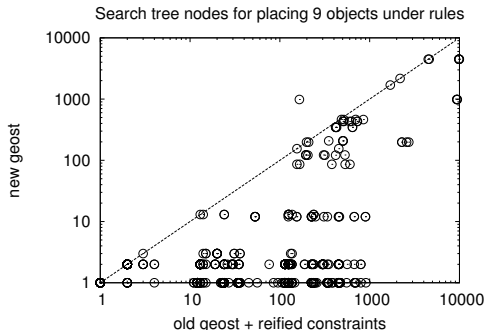
Do Rules Decrease the Search Space?

- ▶ Case study with 9 objects subject to rules (courtesy of Peugeot Citroën):
 - (a) Each object is placed inside a given 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.
- ▶ 600 problem instances were generated by randomly permuting the objects.
- ▶ Each instance was run (1) with the new *geost* constraint and (2) with the earlier implementation expressing constraints (a, b, d, e) with logical combinations of arithmetic constraints.
- ▶ The number of search tree nodes was plotted.

Evaluation of Effectiveness

Results

- ▶ One dot — one instance.
- ▶ Old vs. new implementation.
- ▶ 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.



A Loading-Unloading Problem with Time Windows

Description

- ▶ Place 48 rectangles in a bin.
- ▶ Rectangles enter and leave via the right hand side.
- ▶ Each rectangle has a presence time window.
- ▶ Rectangles must not overlap.
- ▶ When entering and leaving, no other rectangle must be in the way.

A Loading-Unloading Problem with Time Windows

Encoding 1/3

```

start_dur_complete(AllOIDs).
non_overlap(AllOIDs).
visible(AllOIDs, 1, 0).
origin(O, S, D) ---> O^x(D)+S^t(D).
end(O, S, D) ---> O^x(D)+S^t(D)+S^l(D).
start(O) ---> O^x(3).
duration(O) ---> O^x(4).
completion(O) ---> O^x(5).
start_dur_complete(OIDs) --->
  forall(O, objects(OIDs),
    start(O)+duration(O) #= completion(O)).

```

A Loading-Unloading Problem with Time Windows

Encoding 2/3

```
overlap(O, S, Oi, Si, D) --->
  end(O, S, D) #> origin(Oi, Si, D) #/\
  end(Oi, Si, D) #> origin(O, S, D)).

non_overlap(OIDs) --->
  forall(O1, objects(OIDs),
    forall(S1, sboxes([O1^sid]),
      forall(O2, objects(OIDs),
        O1^oid #< O2^oid #=>
          forall(S2, sboxes([O2^sid]),
            #\ (overlap(O1, S1, O2, S2, 1) #/\
              overlap(O1, S1, O2, S2, 2) #/\
              completion(O1) #> start(O2) #/\
              completion(O2) #> start(O1)))))).
```

A Loading-Unloading Problem with Time Windows

Encoding 3/3

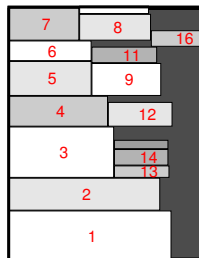
```
visible(OIDs, Dim, Dir) --->
  #\ exists(O, objects(OIDs), masked(OIDs, O, Dim, Dir)).

masked(OIDs, O, Dim, Dir) --->
  exists(Oi, objects(OIDs),
    Oi^oid #\= O^oid #\ masked_by(O, Oi, Dim, Dir)).

masked_by(O, Oi, Dim, Dir) --->
  exists(S, sboxes([O^sid]),
    exists(Si, sboxes([Oi^sid]),
      duration(O) #> 0 #/\
      duration(Oi) #> 0 #/\
      completion(O) #> start(Oi) #/\
      completion(Oi) #> start(O) #/\
      forall(D, [1,2], D #\= Dim #=> overlap(O, S, Oi, Si, D)) #/\
      (Dir #= 0 #=> origin(O, S, Dim) #>= end(Oi, Si, Dim)) #/\
      (Dir #= 1 #=> origin(Oi, Si, Dim) #>= end(O, S, Dim)) #/\
      (start(O) #> start(Oi) #\ completion(O) #< completion(Oi))).
```

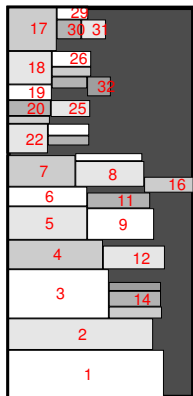
A Loading-Unloading Problem with Time Windows

Solution stage 1: rectangles 1 to 16 enter the bin



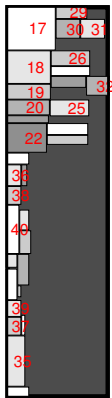
A Loading-Unloading Problem with Time Windows

Solution stage 2: rectangles 17 to 32 enter the bin



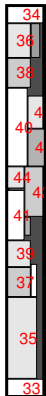
A Loading-Unloading Problem with Time Windows

Solution stage 3: rectangles 1 to 16 leave the container and are replaced by rectangles 33 to 48



A Loading-Unloading Problem with Time Windows

Solution stage 4: after the exit of rectangles 17 to 32, rectangles 33 to 48 are the only rectangles left in the bin



Perspectives

- ▶ Currently, we can only solve for origin variables $o.x[d]$ and shape id $o.sid$, but we would like to allow **more variable attributes**. This implies an m -dimensional **solution** space instead of a k -dimensional **placement** space.
- ▶ The **general** rule mechanism should coexist with **built-in, specialized** methods e.g. for *non-overlapping*.
- ▶ All methods have a **common interface**: they generate forbidden regions, which a sweep algorithm aggregate and use for pruning.
- ▶ A standard encoding of the **time dimension** is urgently needed.

Theoretical Properties

- ▶ Rewriting system is confluent and terminating.
- ▶ Size bound on generated indexicals is known.
- ▶ Pathological cases can be constructed.
- ▶ CSE plays an important role in preventing code size from exploding.

Conclusion

- ▶ We have presented a global constraint over k -dimensional objects and shapes subject to rules written in a language based on arithmetic and first-order logic.
- ▶ We have shown how to:
 1. rewrite the rules to QFPA formulas,
 2. compile such formulas to procedural indexicals,
 3. aggregate forbidden regions generated by such indexicals by a sweep algorithm for filtering domain variables.
- ▶ We have begun to explore a way to compile and run efficiently QFPA, a language suitable for problems outside the packing and placement domain.

References



N. Beldiceanu, M. Carlsson, E. Poder, R. Sadek, 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.



M. Carlsson, N. Beldiceanu, J. Martin.

A generic constraint over k -dimensional objects and shapes subject to business rules.

SICS Technical Report T2008:04, 2008.



F. Fages and J. Martin.

From rules to constraint programs with the `Rules2CP` modelling language. Research Report RR-6495, INRIA Rocquencourt, 2008.