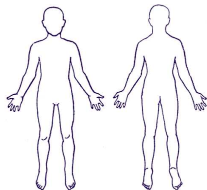
Constraint Programming for Packing Problems

Mats Carlsson, SICS

SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

February 9, 2009

Outline



Constraint Programming

- Introduction

Packing Problems

- Pure Packing Problems
- Packing Problems with Side Constraints

Constraint Programming for Packing Problems

- The *geost* Constraint
- The *geost* Rule Language

Conclusion

- Contributions
- References

Sudoku

In a Nutshell

- Fill all cells so that every row, column and box contains one of $1, \ldots, 9$.

			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

Sudoku
Constraints

- Fill all cells so that every row, column and box contains one of $1, \ldots, 9$.

			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

Sudoku
Constraints

- Fill all cells so that every row, column and box contains one of $1, \ldots, 9$.

			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

A Task for Constraint Programming
Constraints

- Fill all cells so that every row, column and box contains one of $1, \ldots, 9$.

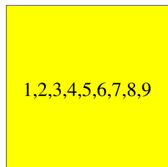
			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

Sudoku

Domains and Propagation

- Remove values incompatible with every row, column and box containing one of $1, \ldots, 9$.

			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

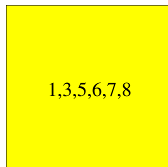


Sudoku

Domains and Propagation

- Remove values incompatible with every row, column and box containing one of $1, \ldots, 9$.

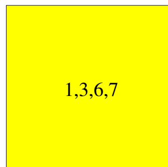
			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			



Sudoku
Domains and Propagation

- Remove values incompatible with every row, column and box containing one of $1, \ldots, 9$.

			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

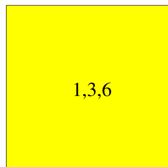


Sudoku

Domains and Propagation

- Remove values incompatible with every row, column and box containing one of $1, \ldots, 9$.

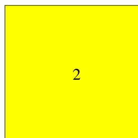
			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			



Sudoku
Solving

- More clever propagation is possible (see figure).
- Iterate propagation until fixpoint.
- If still not solved, guess some value, and iterate again.
- A well-designed Sudoku puzzle can be solved without guessing.

			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			



Sudoku

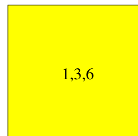
Constraint Programming Model

Variables Every empty cell

Domains 1, . . . , 9 for all empty cells

Constraints *alldifferent* (27 of them)

			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			



Kakuro

Constraint Programming Model

Variables	Every empty cell
Domains	$1, \ldots, 9$ for all cells
Constraints	*alldifferent* for every row and column; *sum* for every row and column

Constraint Satisfaction Problem (CSP)
Definition

A *Constraint Satisfaction Problem* is a triple

$$(Z, D, C)$$

where:

- Z is a finite set of variables $x_1, x_2, \ldots, x_n$,
- D is a function that maps every variable $x_i \in Z$ to a set of integers, $D(x_i)$, the *domain* of x_i, the set of values that x_i can take,
- C is a finite set of constraints on subsets of Z. $c(x_1, x_2, \ldots, x_k)$ where $c \in C$ restricts the combination of values that $x_1, x_2, \ldots, x_k$ can take.

Solution Tuple of CSP
Definition

A *Solution Tuple of CSP*

$$(Z, D, C)$$

where

$$Z = (x_1, x_2, \ldots, x_n)$$

is a tuple of integers $v_1, v_2, \ldots, v_n$ such that:

- $\forall 1 \leq i \leq n : v_i \in D(x_i)$
- $\forall c \in C : (x_1 = v_1, \ldots, x_n = v_n)$ satisfies c

Constraint Programming System

Software Architecture

- Constraint store: variables and their domains.
- Propagators (aka. filtering algorithms):
  - Algorithms that impose necessary conditions by removing inconsistent values from the domains.
  - Coroutines that *wake up* when domains change and *prune* other domains — *propagation*.
  - Communicate through shared variables only.

$$\mathit{alldifferent}(\{1,2\},\{2,3\},\{1,3\},\{1,4,5\},\{2,4,6\},\{3,5,6\},\{1\ldots 6,7\})$$

$$\Downarrow$$

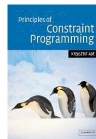
$$\mathit{alldifferent}(\{1,2\},\{2,3\},\{1,3\},\{\not 1,4,5\},\{\not 2,4,6\},\{\not 3,5,6\},\{\not 1\ldots \not 6,7\})$$

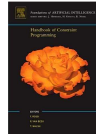
Global Constraints

They have many facets

- For example, *alldifferent*, *sum*.
- Capture relations on a non-fixed number of variables.
- Capture recurring abstractions in constraint models from major application domains.
- Are equipped with propagators that may use graph theory, flow theory, formal language theory, computational geometry, dynamic programming, . . .
- Tend to be vendor specific: no standard exists.
- The Global Constraint Catalogue lists 313 global constraints from the literature and from existing systems giving semantics in terms of graph properties and/or finite automata.

Constraint Programming
The CP Community





CP 2009

SweConsNet

Packing Problems

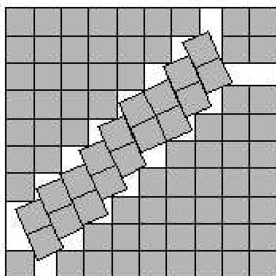
Problem Classification
Some Common Problems

- k-dimensional Bin Packing (usually $k \leq 3$)
  - Minimize the no. of bins required to pack all items.
  - Rotations: none/orthogonal/all.
- k-dimensional Pallet Loading (usually $k = 2$)
  - All items have the same shape (modulo rotation).
  - Maximize the number of items on the pallet.
- Bin design
  - Find the best shape of bins in order to pack a given set of items.
- Item design
  - Find the best shape of items in order to fill a given placement space.

2-dimensional Bin Packing

An optimal solution

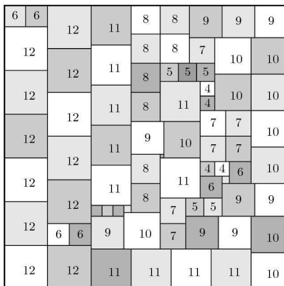
Pack 86 squares into a fixed placement space with free rotation.



2-dimensional Bin Packing

A solution

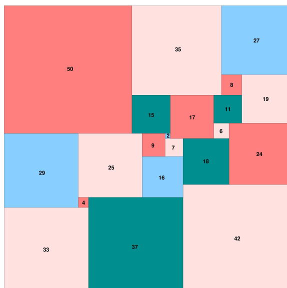
Pack 1 square of size 1, 2 squares of size 2, . . . , 12 squares of size 12 into a large square of size 78.



2-dimensional Bin Packing

A solution

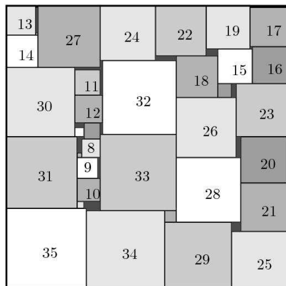
Pack squares of size 2, 4, 6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 24, 25, 27, 29, 33, 35, 37, 42, 50 into a square of size 112.



2-dimensional Bin Design

An optimal solution

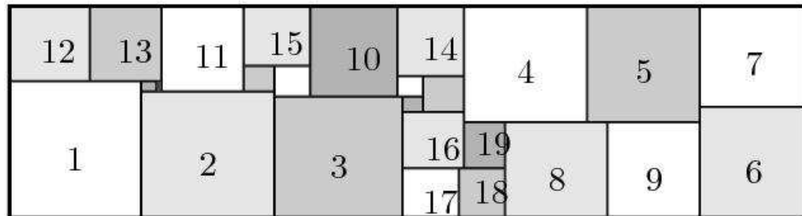
Pack squares of size $1, 2, \ldots, 35$ into a square with minimal area.



2-dimensional Bin Design

An optimal solution

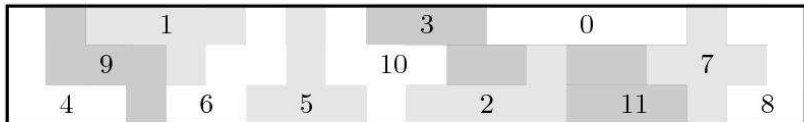
Pack rectangles $1 \times 2, \ldots, 26 \times 27$ with orthogonal rotation into a rectangle with minimal area.



2-dimensional Bin Packing

A solution

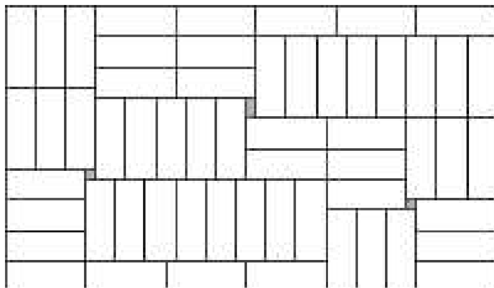
Tile a rectangle with 12 distinctly shaped pentominoes with orthogonal rotation and flipping.



2-dimensional Pallet Loading

An optimal solution to (49,28,8,3)

Fit as many 8×3 items as possible onto a 49×8 pallet with orthogonal rotation. The optimal value is 57.

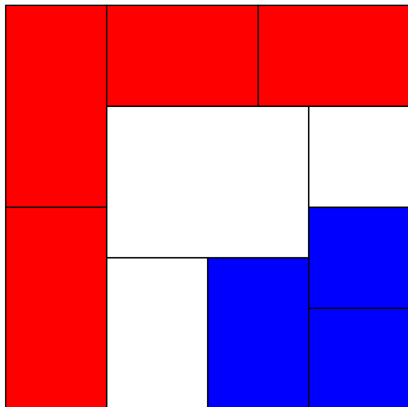


Packing Problems with Side Constraints

- Vessel loading.
  - 2D bin packing + some pairs of items must be separated by a minimal distance.
- Floor planning.
  - 2D/3D item design with size bounds.
  - Topological constraints: adjacency, visibility.
- Chip Design.
- Packing for transportation.
  - Gravity, weight balancing, fragility, stability, ...
  - Company-specific rules.

Vessel Loading
A solution

- Place objects of size 2×4, 2×4, 2×3, 2×3, 2×3, 2×2, 2×2, 3×4, 2×3, 2×2 onto an 8×8 floor.
- Red and blue objects must be at least 2 units apart.
- Problem 8 of `http://www.csplib.org`

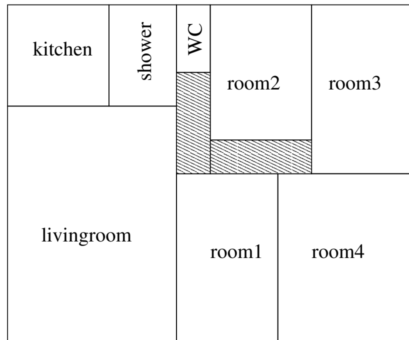


Floor Planning

An optimal plan of 12 rooms on a 12×10 floor

Room	Area	Lmin	Wmin	Ori
Room 1–3	[11, 15]	3	3	N or S
Room 4	[15, 20]	3	3	S
Livingroom	[33, 42]	4	4	S or W
Kitchen	[9, 15]	3	3	N or S
Shower	[6, 9]	2	2	
WC	[1, 2]	1	1	
Corridor 1–2	[1, 12]	1	1	

- All rooms except the kitchen abut a corridor.
- The kitchen abuts the livingroom and the shower.
- The WC abuts the kitchen or the shower.
- Minimize the total corridor area.



Chip Design
Problem Statement

Input

1. A rectangular area — the floorplan.
2. A set of rectangles — the blocks.
3. Some blocks have a fixed position, others are movable.
4. A list of pins per block, with fixed relative locations.
5. A list of nets. Each net is a subset of pins, all of which must be wired together.

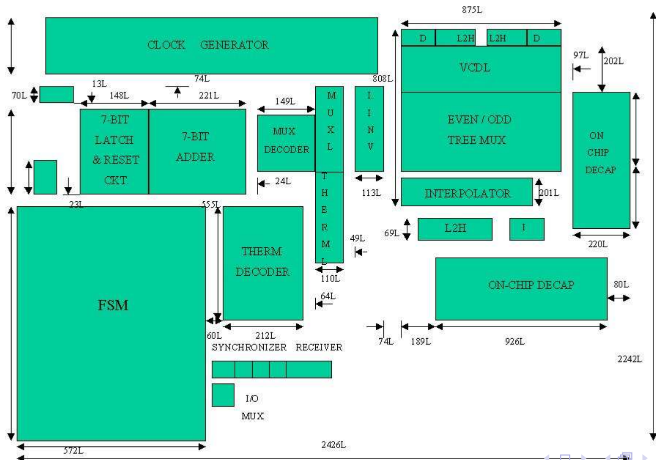
Cost of a net

The perimeter length of the bounding box defined by the pins.

The problem

Place all blocks within the floorplan without overlap, minimizing the total cost.

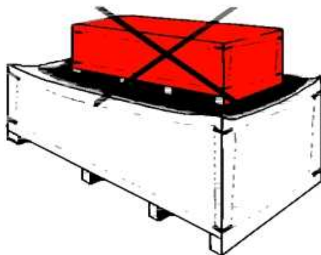
Chip Design
A Solution



A Packing Rule from the Real World

Industry State of the Art

"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 ensembles."



The NET-WMS Idea

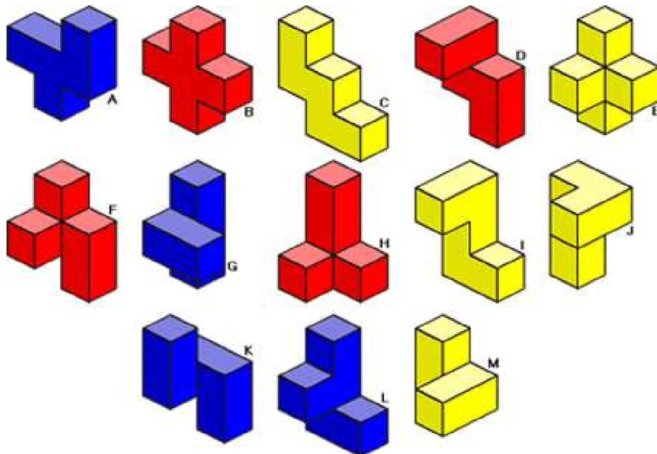
Formal Packing Rules are a Good Thing

With a formal packing rule language, we can come up with useful software for tasks like:

- generating pickup-delivery schedules wrt. loading-unloading placement constraints
- load balancing
- minimizing the number of containers
- filling a pallet for optimal stability or space usage

Constraint Programming for Packing Problems

Can you pack these 13 pieces into a $4 \times 4 \times 4$ cube?



A Global Constraint for Packing

The *geost* Constraint

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

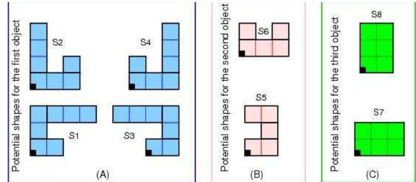
- k Number of dimensions.
- $\mathcal{O}$ Set of objects o with unique *object id* $o.oid$ (an integer), *shape id* $o.sid$, *origin* $o.x[d]$, $1 \leq d \leq k$, *optionally more* integer attributes.
- $\mathcal{S}$ Set of shifted boxes s with *shape id* $s.sid$, *shift offset* $s.t[d]$, $1 \leq d \leq k$, *size* $s.l[d] (s.l[d] > 0, 1 \leq d \leq k)$, *optionally more* attributes (all integers).
- $\mathcal{R}$ Set of rules, referring to the above attributes.
- Semantics: Ground instance is true iff it satisfies the rules.

geost Example

3 non-overlapping objects, 8 shapes: problem

```
geost(2,
  [object(1,S1,[X1,Y1]),
   object(2,S2,[X2,Y2]),
   object(3,S3,[X3,Y3])],
  [sbox(1,[0,0],[2,1]),
   sbox(1,[0,1],[1,2]),
   sbox(1,[1,2],[3,1]),
   sbox(2,[0,0],[3,1]),
   sbox(2,[0,1],[1,3]),
   sbox(2,[2,1],[1,1]),
   sbox(3,[0,0],[2,1]),
   sbox(3,[1,1],[1,2]),
   sbox(3,[2,2],[3,1]),
   sbox(4,[0,0],[3,1]),
   sbox(4,[0,1],[1,1]),
   sbox(4,[2,1],[1,3]),
   sbox(5,[0,0],[2,1]),
   sbox(5,[1,1],[1,1]),
   sbox(5,[0,2],[2,1]),
   sbox(6,[0,0],[3,1]),
   sbox(6,[0,1],[1,1]),
   sbox(6,[2,1],[1,1]),
   sbox(7,[0,0],[3,2]),
   sbox(8,[0,0],[2,3])],
  [nonoverlapping([1,2,3])])
```

$X_1, X_2, X_3 \in 1..5,\ Y_1, Y_2, Y_3 \in 1..4$
$S_1 \in 1..4,\ S_2 \in 5..6,\ S_3 \in 7..8$

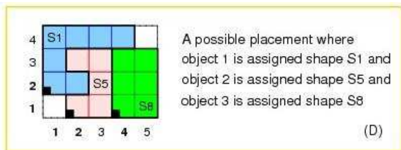


geost Example

3 non-overlapping objects, 8 shapes: a solution

```
geost(2,
  [object(1,S1,[X1,Y1]),
   object(2,S2,[X2,Y2]),
   object(3,S3,[X3,Y3])],
  [sbox(1,[0,0],[2,1]),
   sbox(1,[0,1],[1,2]),
   sbox(1,[1,2],[3,1]),
   sbox(2,[0,0],[3,1]),
   sbox(2,[0,1],[1,3]),
   sbox(2,[2,1],[1,1]),
   sbox(3,[0,0],[2,1]),
   sbox(3,[1,1],[1,2]),
   sbox(3,[2,2],[3,1]),
   sbox(4,[0,0],[3,1]),
   sbox(4,[0,1],[1,1]),
   sbox(4,[2,1],[1,3]),
   sbox(5,[0,0],[2,1]),
   sbox(5,[1,1],[1,1]),
   sbox(5,[0,2],[2,1]),
   sbox(6,[0,0],[3,1]),
   sbox(6,[0,1],[1,1]),
   sbox(6,[2,1],[1,1]),
   sbox(7,[0,0],[3,2]),
   sbox(8,[0,0],[2,3])],
  [nonoverlapping([1,2,3])])
```

$$X_1 = 1, Y_1 = 2$$
$$X_2 = 2, Y_2 = 1$$
$$X_3 = 4, Y_3 = 1$$
$$S_1 = 1, S_2 = 5, S_3 = 8$$



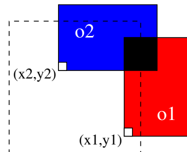
geost Propagation
Forbidden regions

Forbidden region for object o wrt. formula F

A nonempty k-dimensional hyperrectangle (orthotope) r such that if r contains the origin of o, F is necessarily false.

geost Propagation

Computing forbidden regions for o_1 wrt. $\mathit{nonoverlapping}(o_1, o_2)$



Let o_1, o_2 be rectangles with origin $(x_1, y_1) \in \mathbb{Z}^2, (x_2, y_2) \in \mathbb{Z}^2$ and of size $(w_1, h_1) \in \mathbb{Z}^2_+, (w_2, h_2) \in \mathbb{Z}^2_+$. We have:

$$\begin{array}{rl} & \neg \mathit{nonoverlapping}(o_1, o_2) \Leftrightarrow \\ & x_1 + w_1 > x_2 \wedge x_2 + w_2 > x_1 \wedge y_1 + h_1 > y_2 \wedge y_2 + h_2 > y_1 \Leftrightarrow \\ (x_1, y_1) & \in \{(x, y) \in \mathbb{Z}^2 \mid x > x_2 - w_1\} \\ & \cap \{(x, y) \in \mathbb{Z}^2 \mid x < x_2 + w_2\} \\ & \cap \{(x, y) \in \mathbb{Z}^2 \mid y > y_2 - h_1\} \\ & \cap \{(x, y) \in \mathbb{Z}^2 \mid y < y_2 + h_2\} \end{array}$$

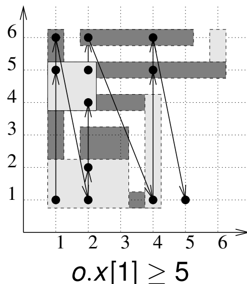
If o_2 is not fixed, we take the intersection of all forbidden regions:

$$\begin{array}{rl} (x_1, y_1) & \in \{(x, y) \in \mathbb{Z}^2 \mid x > \overline{x_2} - w_1\} \\ & \cap \{(x, y) \in \mathbb{Z}^2 \mid x < \underline{x_2} + w_2\} \\ & \cap \{(x, y) \in \mathbb{Z}^2 \mid y > \overline{y_2} - h_1\} \\ & \cap \{(x, y) \in \mathbb{Z}^2 \mid y < \underline{y_2} + h_2\} \end{array}$$

geost Propagation

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

- To adjust the lower bound of $o.x[1]$ for $o \in \mathcal{O}$:
  1. Seek the *lexicographically smallest position* (x', y') that is outside all forbidden regions for o wrt. $\mathcal{R}$.
  2. $o.x[1] \geq x'$.
- Similarly for upper bounds, all dimensions, all objects, until fixpoint.



$o.x[1] \geq 5$

geost and Rules

Some Observations

- We compute not forbidden regions, but *generators of forbidden regions as functions of the constraint store*.
- Forbidden region generators can be *automatically computed* from a large class of formulae.
- We can capture this class of formulae with a *formal language*.
- Side constraints in packing problems can be expressed:
  1. either as constraints in conjunction with *geost* (the sweep is unaware of them),
  2. or as rules given to *geost* in $\mathcal{R}$ (the sweep avoids positions that violate them).
- *Alternative 2 generally leads to better propagation (less search).*

geost and Rules

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

- $\mathcal{R}$ is a set of statements in a first order logic based rule language containing arithmetic constraints.
- Two layer language design.
  - The *full language* has many features and is rewritten into...
  - The *core fragment*, a subset of Quantifier-Free Presburger Arithmetic (QFPA). Small, clear semantics, amenable to compilation.

Full Language (1 of 3)

$$
\begin{array}{lcll}
\textit{sentence} & ::= & \textit{macro} \mid \textit{fol} & \\
\textit{macro} & ::= & \textit{term} \Longrightarrow \textit{fol} & \\
 & \mid & \textit{term} \Longrightarrow \textit{expr} & \\
\textit{fol} & ::= & \neg \textit{fol} & \{\text{negation}\} \\
 & \mid & \textit{fol} \wedge \textit{fol} & \{\text{conjunction}\} \\
 & \mid & \textit{fol} \vee \textit{fol} & \{\text{disjunction}\} \\
 & \mid & \textit{fol} \Rightarrow \textit{fol} & \{\text{implication}\} \\
 & \mid & \textit{fol} \Leftrightarrow \textit{fol} & \{\text{equivalence}\} \\
 & \mid & \exists(\textit{var}, \textit{collection}, \textit{fol}) & \{\text{existential quantification}\} \\
 & \mid & \forall(\textit{var}, \textit{collection}, \textit{fol}) & \{\text{universal quantification}\} \\
 & \mid & \#(\textit{var}, \textit{collection}, \textit{integer}, \textit{integer}, \textit{fol}) & \{\text{cardinality}\} \\
 & \mid & \texttt{true} & \\
 & \mid & \texttt{false} & \\
 & \mid & \textit{expr} \quad \textit{relop} \quad \textit{expr} & \{\text{over } \mathbb{Q}\} \\
 & \mid & \textit{term} & \{\text{macro application}\}
\end{array}
$$

Full Language (2 of 3)

expr	::=	$expr + expr$	
	|	$expr - expr$	
	|	$\min(expr, expr)$	
	|	$\max(expr, expr)$	
	|	$expr \times conexpr$	
	|	$conexpr \times expr$	
	|	$expr / conexpr$	
	|	*attref*	
	|	*integer*	
	|	@(*var*, *collection*, *op*, *expr*, *expr*)	{folding}
	|	*variable*	{quantified variable}
	|	*term*	{macro application}
attref	::=	*entity*.*attr*	{of object or sbox}
conexpr	::=	*expr*	{where *expr* is ground}

Full Language (3 of 3)

attr	::=	*term*	{attribute name}
	|	*variable*	{quantified variable}
relop	::=	$< \mid = \mid > \mid \neq \mid \leq \mid \geq$	
op	::=	$+$ | min | max	
collection	::=	*list*	
	|	`objects`(*list*)	{list of object ids}
	|	`sboxes`(*list*)	{list of shape ids}
list	::=	[]	
	|	[*term*|*list*]	

Full Language Features

- The macro *head*$\Longrightarrow$*body* provides an abbreviation for *body* occurring anywhere; any instance of *head* is replaced by the corresponding instance of *body*.
- $\Rightarrow, \Leftrightarrow, \exists, \forall, \#$, expanding to $\neg, \wedge, \vee$.
- Cardinality operator $\#$, folding operator `@`.

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

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

Full Language Example
Vessel Loading

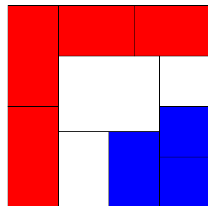
“Red and blue objects must be at least 2 units apart.”

```
origin(O1,S1,D) ---> O1^x(D)+S1^t(D)).

end(O1,S1,D) ---> O1^x(D)+S1^t(D)+S1^l(D).

tooclose(O1,O2,S1,S2,D) --->
   end(O1,S1,D)+2 #> origin(O2,S2,D) #/\
   end(O2,S2,D)+2 #> origin(O1,S1,D).

apart(O1,O2) --->
   forall(S1,sboxes([O1^sid]),
       forall(S2,sboxes([O2^sid]),
           #\ tooclose(O1,O2,S1,S2,1) #\/
           #\ tooclose(O1,O2,S1,S2,2))).
```



Full Language Example
Floor Planning

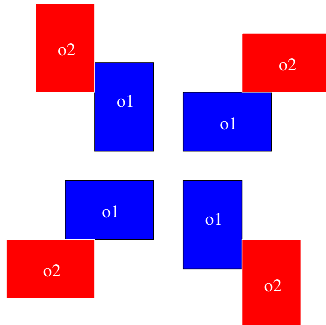
“Rooms O_1 and room O_2 abut.”

```
origin(O1,S1,D) ---> O1^x(D)+S1^t(D).

end(O1,S1,D) ---> O1^x(D)+S1^t(D)+S1^l(D).

overlap(O1,S1,O2,S2,D) --->
   end(O1,S1,D) #> origin(O2,S2,D) #/\
   end(O2,S2,D) #> origin(O1,S1,D).

abut(O1,O2) --->
   exists(S1,sboxes([O1^sid]),
       exists(S2,sboxes([O2^sid]),
           ((origin(O1,S1,1) #= end(O2,S2,1) #\/
             origin(O2,S2,1) #= end(O1,S1,1)) #/\
            overlap(O1,S1,O2,S2,2)) #\/
           ((origin(O1,S1,2) #= end(O2,S2,2) #\/
             origin(O2,S2,2) #= end(O1,S1,2)) #/\
            overlap(O1,S1,O2,S2,1))).
```



Core Fragment

Subset of Quantifier-Free Presburger Arithmetic (QFPA).

$$
\begin{array}{lcll}
\mathit{qfpa} & ::= & \mathit{qfpa} \wedge \mathit{qfpa} & \{\text{conjunction}\} \\
 & | & \mathit{qfpa} \vee \mathit{qfpa} & \{\text{disjunction}\} \\
 & | & \sum_i \mathit{integer}_i \cdot \mathit{attref}_i \geq \mathit{integer} & \{\text{integer linear inequation}\} \\
 & & & \\
\mathit{attref} & ::= & \mathit{entity}.\mathit{attr} & \{\text{nonground reference}\}
\end{array}
$$

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 subformulae.

Computing FR Generators

Mapping QFPA q and Object o to FR Generator $F(q,o)$

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

where $\mathrm{MAX}(f)$ replaces x by $\overline{x}$ resp. $\underline{x}$ in f.

geost Propagation Revisited

Lexicographic Sweep for *geost*$(k, \mathcal{O}, \mathcal{S}, \mathcal{R})$

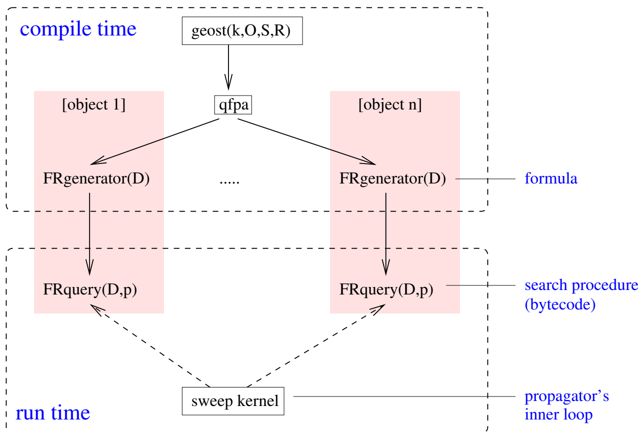
- To adjust the lower bound of $o.x[1]$ for $o \in \mathcal{O}$:
  1. Seek the lexicographically smallest position (x', y') that is outside all forbidden regions for o wrt. $\mathcal{R}$.
  2. $o.x[1] \geq x'$.
- Similarly for upper bounds, all dimensions, all objects, until fixpoint.

Please note:

- The key operation is a *query* “does there exist a FR that contains the sweep position?”
- FR generators are generally nested $\cap/\cup$ formulae; the number of FRs can blow up exponentially.
- Instead of computing them all, it is better to *search* for a FR that contains the sweep position. Treat $\cup$ as a branching point.

Propagator Architecture

Lexicographic Sweep for *geost*$(k, \mathcal{O}, \mathcal{S}, \mathcal{R})$



Theoretical Properties

- Rewriting system is confluent and terminating.
- Size bound on FR generators is known.
- Pathological cases can be constructed.
- Common subexpression elimination helps prevent code size explosion.

Conclusion

Contributions

Constraint Programming for Packing Problems

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

- A new 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 formulae,
  2. compile QFPA formulae to forbidden region generators, which are functions of the constraint store,
  3. propagate *geost* with a sweep algorithm that systematically seeks positions that are outside all forbidden regions computed by such functions.
- We have begun to explore a way to efficiently compile and execute QFPA, a language suitable for problems outside the packing and placement domain.

References

N. Beldiceanu, M. Carlsson, J.-X. Rampon.
Global Constraint Catalogue.
SICS Technical Report T2005:08, 2005.
http://www.emn.fr/x-info/sdemasse/gccat/

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.
Proc. CP 2007, *LNCS* 4741, pages 180–194. Springer.

N. Beldiceanu, M. Carlsson, E. Poder.
New Filtering for the *cumulative* Constraint in the Context of Non-Overlapping Rectangles.
Proc. CPAIOR 2008, *LNCS* 5105, pages 21–35. Springer.

M. Carlsson, N. Beldiceanu, J. Martin.
A generic constraint over *k*-dimensional objects and shapes subject to business rules.
Proc. CP 2008, volume 5202 of *LNCS* 5202, pages 220–234. Springer.