A Cardinal Set Representation for MiniZinc

Sullivan Bitho

Internship supervisors:
Guido Tack
Jip J. Dekker

September 6, 2023

MONASH University

Nantes Université

The Social Golfers Problem

[1]

Figure: A group of 16 friends wants to organize a golf tournament.

- 4 teams of 4 golfers would play over 3 rounds.
- For every round:
  - Each golfer can only play once per round.
  - Each golfer must play with golfers he has never played with before.

[1] Picture from harbourridge.com.

A Social Golfers model formulated with sets

- A $team$ is a *set* of 4 golfers, a $round$ is an integer in $1..3$.
- **Decision:** Teams of golfers per round.
  An array of $[round \times team]$: $round\_team\_golfers$
- **Constraints:**
  - Each golfer can only play once per round:
    $\forall r \in round : |\bigcap_{t \in team} round\_group\_golfers_{r,t}| = 0$
  - Each golfer must play with golfers he has never played with before:
    $\forall r_1 < r_2 \in round, \forall t_1, t_2 \in team$:
    $|round\_group\_golfers_{r_1,t_1} \cap round\_group\_golfers_{r_2,t_2}| \leq 1$

A Social Golfers model formulated without sets

- A $team$ is an *array* of 4 golfers, a $round$ is an integer in $1..3$.
- **Decision:** Teams of golfers per round.
  An array of $[round \times team]$: $round\_team\_golfers$
- **Constraints:**
  - Each golfer can only play once per round:
    $\forall r \in round, \forall i \in 1..(|round\_group\_golfers_r| - 1),$
    $\forall j \in (i+1)..|round\_group\_golfers_r|$:
    $\texttt{concatene\_all}(round\_group\_golfers_r)_i \neq$
    $\texttt{concatene\_all}(round\_group\_golfers_r)_j$
  - Each golfer must play with golfers he has never played with before:
    $\forall r_1 < r_2 \in round, \forall t_1, t_2 \in team, \forall g \in golfers,$:
    $\texttt{occurence}(g, ($
    $\texttt{concatene}(round\_group\_golfers_{r_1,t_1}, round\_group\_golfers_{r_2,t_2})$
    $)) \leq 1$

Automatic set encoding

- Set models feels more natural and expressive.
- Many solvers on the market do not feature making decisions about sets.
- Can we automate set encodings for those solvers? How efficient would it?
- We propose a library that automatically encode sets.

Contents

Table of Contents

Constraint Programming

- *Constraint Programming* (*CP*)[2]: programming paradigm aimed at solving hard combinatorial problems.
- *Constraint Satisfaction Problem* (*CSP*)[3]: satisfy a set of constraints.
- CSP solving techniques:
  - tree search, backtracking...
  - constraint propagations, consistency techniques...

[2] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[3] Khaled Ghedira. *Constraint satisfaction problems: csp formalisms and techniques*. John Wiley & Sons, 2013.

Global Constraints

- Global Constraint are powerful high level constraints that can constraint $n \geq 1$ variables *simultaneously*.
- `element`(i, Tab, val):
  val is equal to the i^{th} item of Tab.
- `all_different`$(Vars)$:
  Enforce all variables of the collection $Vars$ to take distinct values.
- `global_cardinality`$(Vars, Vals, Noccurs)$:
  Each value $Vals_i$ (with $i \in 1..|Vals|$) should be taken by exactly $Noccurs_i$ variables of the $Vars$ collection.

$$\texttt{global\_cardinality}(\underbrace{[3,3,8,6]}_{Vars}, \underbrace{[3,5,6]}_{Vals}, \underbrace{[2,0,1]}_{Noccurs})$$

Willem-Jan van Hoeve and Irit Katriel. “Global constraints”. In: *Foundations of Artificial Intelligence*. Vol. 2. Elsevier, 2006, pp. 169–208.

Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. *Global constraint catalog, (revision a)*. 2012.

MiniZinc

- Constraint modelling language (free and open source)
- Declarative language
- Solver-independant

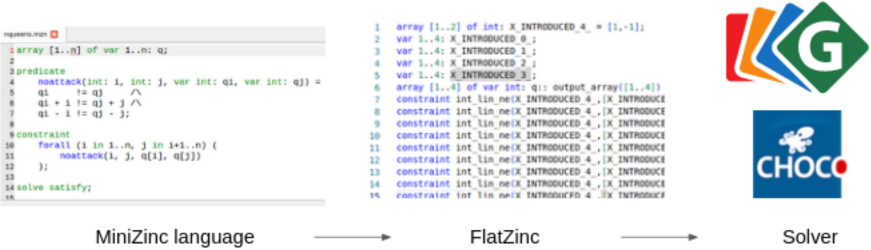


Figure: MiniZinc

Set variable

Definition

A set domain D is a set of sets containing all possible set values of a set variable s.

A set variable is decided over its set domain D.

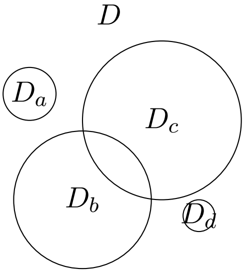


Figure: A visualization of a set domain D with its set elements D_e

Conjunto set representation

Definition

A *set interval* I defines a lattice of sets, partially ordered by set inclusion, such that $I = [glb, lub]$, with $glb = \bigcap_{i=1}^{n} D_i$ the intersection of all sets inside D, and $lub = \bigcup_{i=1}^{n} D_i$ the union of all sets inside D.

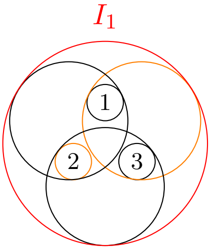
Definition

The *cardinality* C of a set variable s ranges in the integer domain $C_{min}..C_{max}$, where $C_{min} = card(glb(s))$ and $C_{max} = card(lub(s))$.

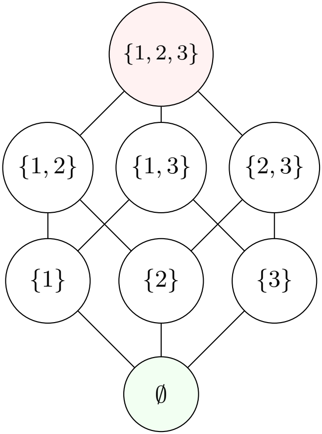
Conjunto[4] represents a set by a greatest lower bound glb, a least upper bound lub, a lower bound cardinality C_{min}, an upper bound cardinality C_{max}.

[4] Carmen Gervet. "Conjunto: constraint logic programming with finite set domains". In: *ILPS*. 1994.

Conjunto set representation



(a) Set variable s_1 range over I_1, the convex enclosure of D_1



(b) Lattice representation of I_1

Figure: Two different vizualisations of I_1

Boolean set encoding

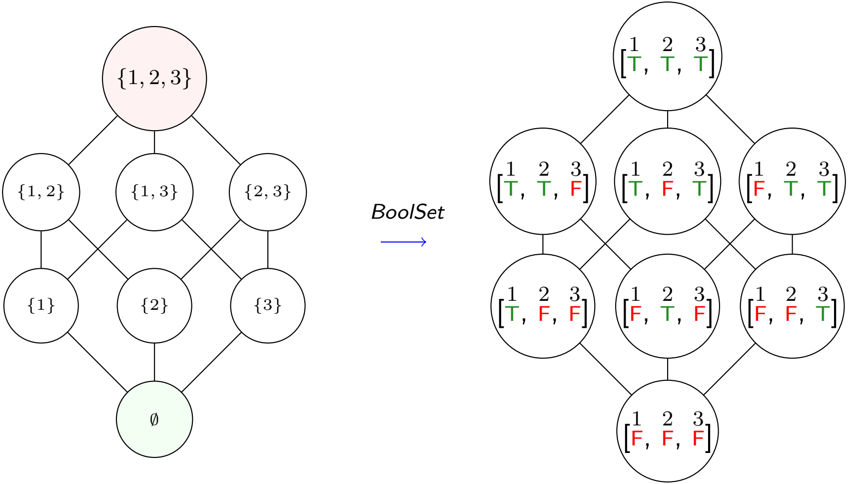
- Array representation: breaks symmetric set representation
- Let s be a set variable with a domain D. Let f_B be the function which encode a set into an array of Boolean variables. Let f_B^{-1} the reverse function of f_B. xb is the array of Boolean variables encoding s such that:

$$
\begin{aligned}
f_B(s) &= x \\
f_B^{-1}(xb) &= s \\
xb_i &= true \quad \Leftrightarrow i \in s, \forall i \in 1..card(s) \\
xb_i &= false \ \Leftrightarrow i \notin s, \forall i \in 1..card(s)
\end{aligned}
$$

- Encoding name: *BoolSet*

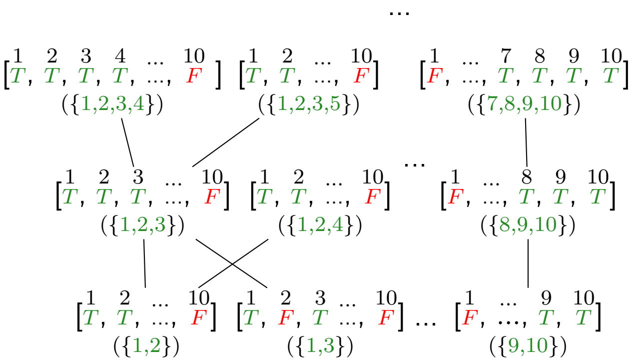
BoolSet encoding: Example 1

Let consider a set s_2 with a domain $D_2 = \mathcal{P}(\{1,2,3\})$ and a variable cardinality $C_2 \in \{0,1,2,3\}$ ($C_{2_{min}} = 0, C_{2_{max}} = 3$)



BoolSet encoding: Example 2

Let consider a set s_3 with a domain $D_3 = \mathcal{P}(\{1,2,3,4,5,6,7,8,9,10\})$ and a variable cardinality $C_3 \in \{2,4\}$ ($C_{3_{min}} = 2, C_{3_{max}} = 4$)



- BoolSet requires to create an array containing 10 Boolean variables.
- CardSet, only needs C_max variables.

Table of Contents

Cardinality set encoding

- Given a set variable s with a domain D and C_{min} and C_{max} respectively the minimum and maximum possible cardinality of s. f_C is the function that encodes a set into an array of integer variables. f_C^{-1} is the reverse function of f_C. Let y be the array of integers encoding s. We define $m = C_{max} - card(s)$, and d_i, the dummy value for element y_i, such that:

$$f_C(s) = y$$
$$f_C^{-1}(y) = s$$
$$card(s) \in C_{min}..C_{max}$$
$$y_i = min(s \setminus \bigcup_{j=1}^{i-1} s_j), \forall i \in 1..card(s)$$
$$y_j = d_k, \forall j \in (card(s)+1)..C_{max}, \forall k \in 1..m$$

- Encoding name: *CardSet*.

CardSet algorithm

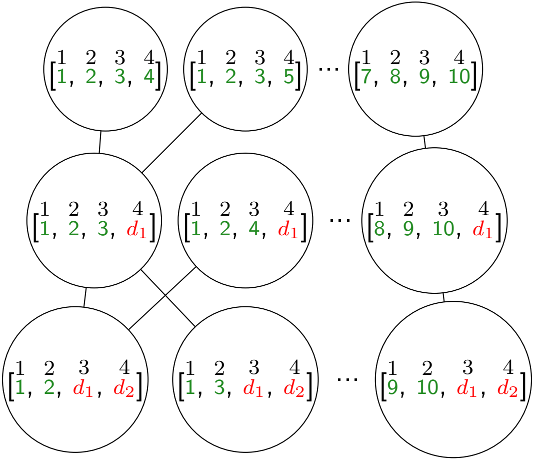
- f_C is named SET2INT.
- The "extra elements" of xi that do not represents any value in s are *dummy* elements.

Algorithm SET2INT

Require: A set s
Ensure: An array of integers xi
1: CONSTRAINT $C = card(s)$
2: CONSTRAINT $C_{min} =$ `lower_bound`(C)
3: CONSTRAINT $C_{max} =$ `upper_bound`(C)
4: CONSTRAINT $dummies$ be the set of the possible dummy elements for xi
5: CONSTRAINT xi be an array such that $xi_j \in$ `upper_bound`$(s) \cup dummies, \forall j \in 1..C_{max}$
6: GLOBAL CONSTRAINT `strictly_increasing`(xi)
7: CONSTRAINT $xi_j \notin dummies, \forall j \in 1..C$
8: CONSTRAINT $xi_j \in dummies, \forall j \in (C+1)..C_{max}$
9: CONSTRAINT $s =$ `reverse_set2int`(xi)

CardSet encoding: Example

Let consider a set s_3 with a domain $D_3 = \mathcal{P}(\{1,2,3,4,5,6,7,8,9,10\})$ and a variable cardinality $C_2 \in \{2,4\}$ $(C_{min} = 2, C_{max} = 4)$



Dummy elements

- Set variable with variable cardinality: cannot declare an array of variable size.
- $C\_max - C$ dummy integers elements outside of D?
- Absent value $<>$ is a MiniZinc built-in feature, suitable for our array representation (properties like $n <<> \wedge <>< n$ are both true, and $n =<>$ is false, will simplifies operations involving multiple sets)
- $C\_max - C$ dummy absent elements?
- Which one is better? Dummy integers create *fake overlaps* but allows us to easily constraint the strictly increasing values. Optional values are not supported by all global constraints but avoid *fake overlaps*.
- Both encoding have been implemented.

Set variable constraints encodings

- Set variable membership
- Set variable less or equal
- Set variable intersection and union

Set variable membership

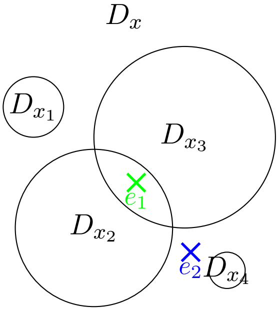
- $e \in f_C^{-1}(xi) \Leftrightarrow \bigvee_{i \in 1..|xi|} xi_i = e$
-

1	2	...	$\lvert xi \rvert + 1$
$xi_1 = e?$	$xi_2 = e?$	...	*True*

- $\texttt{arg\_max}(a)$ return the index of the max value of an array a.

SET__IN

Require: A set s, a value e
Ensure: e belongs to s.
1: CONSTRAINT $xi = \texttt{set2int}(s)$
2: CONSTRAINT $\forall j \in \texttt{index}(xi),\ arr\_b_j = (xi_j == e)$
3: CONSTRAINT $arr\_b_{\texttt{index}(xi)+1} = True$
4: GLOBAL CONSTRAINT $\texttt{arg\_max}(arr_b) \leq card(s)$



(b) Venn Diagram of D_x, with e_1, e_2 two elements

Set variable less or equal

- $s_2 \leq s_2$ means s_1 is *lexicographically* less or equal to s_2.
- GLOBAL CONSTRAINT `lex_lesseq`(u) with u the concatenation of the *CardSet* representation of s_1 and s_2

Algorithm SET_LE

Require: A set s_1, a set of value s_2
Ensure: s_1 is lexicographically less or equal to s_2.

1: **if** $ub(s_1) = \emptyset$ **then**
2: **return** $True$
3: **else if** $ub(s_2) =$ **then**
4: CONSTRAINT $s_2 = \emptyset$
5: **end if**
6: CONSTRAINT $xi =$ `set2int`(s_1)
7: CONSTRAINT $yi =$ `set2int`(s_2)
8: GLOBAL CONSTRAINT `lex_lesseq`(xi, yi)

Set variable intersection and union

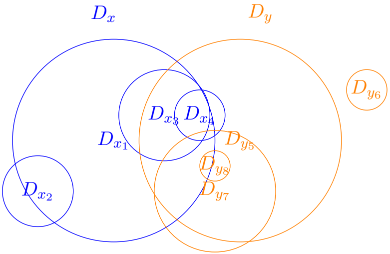
- Let $s_3 = s_1 \cap s_2$, $s_4 = s_1 \cup s_3$
- Counting common elements *noccurs* of $u = (xi, yi)$ with the `global_cardinality` constraint.
- *noccurs*$_i = 2 \Leftrightarrow D_{xy_i} \in s_3$
- *noccurs*$_i \geq 1 \Leftrightarrow D_{xy_i} \in s_4$

SET_INTERSECT

Require: A set s_1, a set s_2

Ensure: Return the set of common values shared by s_1 and s_2

CONSTRAINT $xi =$ `set2int`(s_1)
CONSTRAINT $yi =$ `set2int`(s_2)
Let s_3 be the resulting set with D_3 its domain.
CONSTRAINT $zi =$ `set2int`(s_3)
CONSTRAINT $xyi = [xi, yi]$
Let $noccurs$ be the number of occurence of each value of D_3 in xyi.
GLOBAL CONSTRAINT `global_cardinality`$(xyi, D_3, noccurs)$;
CONSTRAINT $zi_j = D_{3_j}, \forall j : noccurs_j \geq 2$
CONSTRAINT $zi_k \in dummies, \forall k : noccurs_k < 2$
return s_3



(b) Venn Diagram of 2 set domains, D_x and D_y

Global constraints redefinitions

- Disjoint and all disjoint
- Count common element

Disjoint and all disjoint

- `disjoint:`
  - s_1, s_2 two sets of domains D_1, D_2
  - $|s_1 \cap s_2| = 0$
  - $noccurs = \texttt{global cardinality}([xi, yi], D_1 \cup D_2)$
  - $noccurs_i \leq 1, \forall i \in 1..|D_1 \cup D_2|$
- `all_disjoint:`
  - S is a collection of n sets s_i of domains D_i
  - $|\bigcap_{i \in 1..|S|}| = 0$
  - $noccurs = \texttt{global cardinality}($
    $\texttt{array\_union}([xi_k, \forall k \in 1..|S|]), \bigcup_{i \in 1..|S|} D_i)$
  - $noccurs_i \leq 1, \forall i \in 1..|D_1 \cup D_2|$

Count common elements

- **count_common_element**
  - s_1, s_2 two sets of domains D_1, D_2
  - $|s_1 \cap s_2| = n$
  - $noccurs = \texttt{global cardinality}([xi, yi], D_1 \cup D_2)$
  - $\texttt{count}(noccurs, 2)$

Table of Contents

The Steiner's Triple Problem

- Decision variables:
  - $sets$: an array of $nb = (n * (n-1))/6$ set variables ranging from 1 to n.
- Constraints:
  - CONSTRAINT $|sets_i \cap sets_j| \leq 1, \forall i \in 1..nb, \forall j \in i+1..nb$ (1)
  - Symmetrie breaking
    CONSTRAINT $sets_i \geq sets_{i+1}, \forall i \in 1..(nb-1)$ (2)

AD Forbes, Mike J Grannell, and Terry S Griggs. "Steiner triple systems and existentially closed graphs". In: *the electronic journal of combinatorics* 12.1 (2005), R42.

The Steiner's Triple Problem: Experimental results

Table: Steiner's Triple Problem solving times comparison (in seconds).

Instances	Gecode				Chuffed			Highs		
	Set	BoolSet	CardSet		BoolSet	CardSet		BoolSet	CardSet	
			opt	nem		opt	nem		opt	nem
03	0.091	0.119	0.252	0.276	0.09	0.266	0.271	0.091	0.266	0.266
04	0.119	0.097	0.395	0.391	0.099	0.285	0.279	0.112	0.278	0.286
05	0.117	0.257	0.619	0.634	0.603	0.62	0.634	0.143	0.603	0.626
06	0.097	0.134	0.284	0.286	0.125	0.291	0.287	0.838	1.873	1.384
07	0.115	0.144	0.287	0.297	0.157	0.3	0.304	1.815	3.524	4.885
08	3.156	89.951	3.748	3.145	0.433	0.484	0.461	42.653	timeout	timeout
09	0.111	timeout	1.191	2.741	0.378	0.529	0.414	58.834	52.861	136.204
10	timeout	timeout	timeout	timeout	67.626	59.897	64.768	timeout	timeout	timeout
13	timeout	timeout	timeout	timeout	timeout	19.623	3.913	timeout	timeout	timeout
15	0.142	timeout	timeout	timeout	timeout	7.557	timeout	timeout	timeout	timeout
17	0.139	timeout	timeout	timeout	timeout	7.302	timeout	timeout	timeout	timeout

The Social Golfer Problem

- A $team$ is a *set* of 4 golfers, a $round$ is an integer.
- **Decision:** Teams of golfers per round.
  An array of $[round \times team]$: $round\_team\_golfers$
- **Constraints:**
  - Each golfer can only play once per round:
    $\forall r \in round : \bigcap_{t \in team} |round\_group\_golfers_{r,t}| = 0$
  - Each golfer must play with golfers he has never played with before:
    $\forall r_1 < r_2 \in round, \forall t_1, t_2 \in team :$
    $|round\_group\_golfers_{r_1,t_1} \cap round\_group\_golfers_{r_2,t_2}| \leq 1$

Markus Triska. *Solution methods for the social golfer problem*. na, 2008.

The Social Golfer Problem: Experimental results

Table: Social Golfers Problem solving times comparison (in seconds).

Instances	Gecode				Chuffed			Highs		
	Set	BoolSet	CardSet		BoolSet	CardSet		BoolSet	CardSet	
			opt	nem		opt	nem		opt	nem
2_5_4	0.538	180	0.971	0.855	26.187	0.723	0.741	3.904	5.282	4.731
2_8_5	180	180	2.292	2.169	180	2.877	2.95	31.566	29.368	27.801
3_6_4	78.311	180	1.132	0.96	11.852	1.296	1.201	12.303	14.664	11.888
3_6_6	0.974	180	2.27	2.252	180	2.652	2.522	180	180	180
3_7_4	180	180	1.627	1.428	180	2.845	2.134	23.542	24.281	18.407
4_5_4	28.813	180	0.927	1.069	180	1.582	4.102	56.214	69.483	135.641
4_7_4	180	180	2.09	1.841	129.746	2.744	2.693	48.488	85.643	119.109
4_9_4	180	180	4.942	4.459	180	7.357	8.371	80.628	75.274	75.88
5_8_3	180	180	2.05	1.996	180	3.472	2.328	21.215	25.582	28.837
8_5_2	15.689	180	0.668	0.833	2.765	0.695	0.922	7.372	5.542	5.182

Table of Contents

Conclusion

- CardSet is promising
- Global constraint leverage CardSet representation efficiency.
- CardSet_opt is preferable for MiniZinc since absent value is a well implemented "turn-key" type.
- Do not perform better on every problems

Future Work

- CardSet: more benchmarks, library optimization
- Automatic library selection
- Designing efficient propagators for set constraints
- Lazy Clause Generation with set variables.

- Beldiceanu, Nicolas, Mats Carlsson, and Jean-Xavier Rampon. *Global constraint catalog, (revision a)*. 2012.
- Forbes, AD, Mike J Grannell, and Terry S Griggs. "Steiner triple systems and existentially closed graphs". In: *the electronic journal of combinatorics* 12.1 (2005), R42.
- Gervet, Carmen. "Conjunto: constraint logic programming with finite set domains". In: *ILPS*. 1994.
- Ghedira, Khaled. *Constraint satisfaction problems: csp formalisms and techniques*. John Wiley & Sons, 2013.
- Hoeve, Willem-Jan van and Irit Katriel. "Global constraints". In: *Foundations of Artificial Intelligence*. Vol. 2. Elsevier, 2006, pp. 169–208.
- Rossi, Francesca, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.
- Triska, Markus. *Solution methods for the social golfer problem*. na, 2008.

Thank you!

Guido Tack,

Jip J. Dekker,

Peter J. Stuckey,

Evgeny Gurevsky,

Eric Monfroy,

Charles Prud'homme