# Comparing EventB, {log} and Why3 Models of Sparse Sets

Maximiliano Cristiá[1] Catherine Dubois[2]

[1] Universidad Nacional de Rosario and CIFASIS, Argentina
[2] ENSIIE - INRIA - Université Paris-Saclay, LMF, France

# Introduction

Contexte : vérification formelle d'algorithmes et résultats en programmation par contraintes
$\longrightarrow$ Solveur CP(FD) formellement vérifié en Coq [FM 2012, WFLP 2020]

CSP (problème de satisfaction de contraintes) $= (\mathcal{X}, \mathcal{D}, \mathcal{C})$
- $\mathcal{X}$ : ensemble des variables, $\mathcal{D}(x)$ : domaine de la variable $x$, ensemble fini
- $\mathcal{C}$ : ensemble des contraintes à satisfaire

Techniques principales de résolution d'un csp : élagage des domaines + énumération des valeurs + backtracking

Représentation d'un domaine : listes d'intervalles [Coq, JFLA 2019], bitvecteurs, ensembles creux, ...

Contribution : (première) vérification formelle des ensembles creux et des opérations utilisées dans les solveurs CP(FD) (member, remove, singleton) dans 3 formalismes : EventB, {log} et WhyML

# Pourquoi EventB, {log} et WhyML ?

EventB (variante de la méthode B), Rodin : théorie des ensembles, raffinement, obligations de preuve, preuve automatique et interactive, prouveurs dédiés et externes

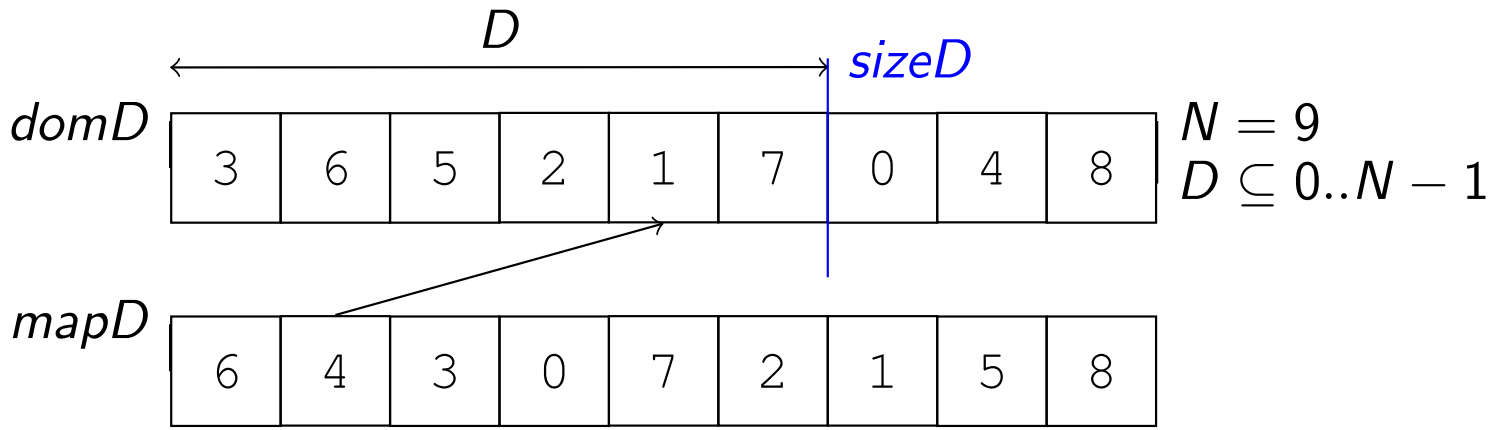
{log} : langage de programmation logique à contraintes, théorie des ensembles, obligations de preuve, procédures de décision

WhyML, Why3 : langage polymorphe à la ML avec assertions (pre, post), bibliothèque, obligations de preuve, preuve automatique et interactive, prouveurs externes

# Ensembles creux

## Structure de données

Représentation d'ensembles finis [Briggs and Torczon 93] (ici sous-ensembles d'un intervalle de $\mathbb{N}$) à l'aide de deux tableaux et un entier




Invariants

$$D \subseteq 0..N-1 \wedge D = \{domD[i] \mid 0 \leq i < sizeD\} \quad (P_1)$$

$$mapD[v] = i \Leftrightarrow domD[i] = v, \text{ for all } i \text{ and } v \quad (P_2)$$

# Ensembles creux

## Opérations considérées

# Ensembles creux

## Opérations considérées

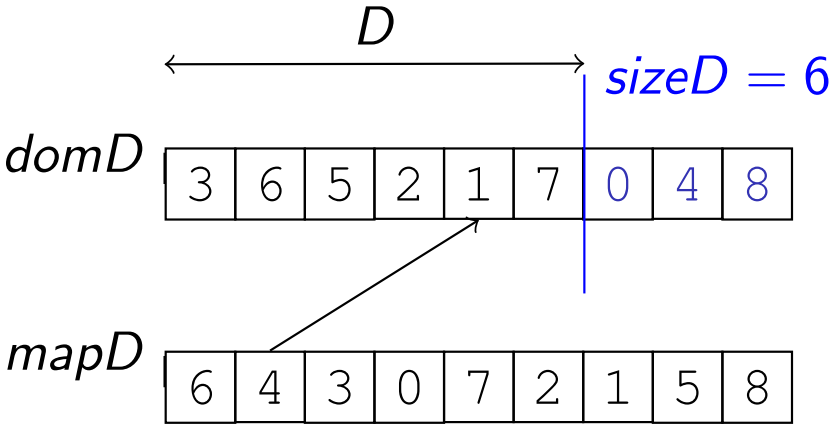
- $v \in D\,? \longrightarrow mapD[v] < sizeD\,? \longrightarrow$ en temps constant

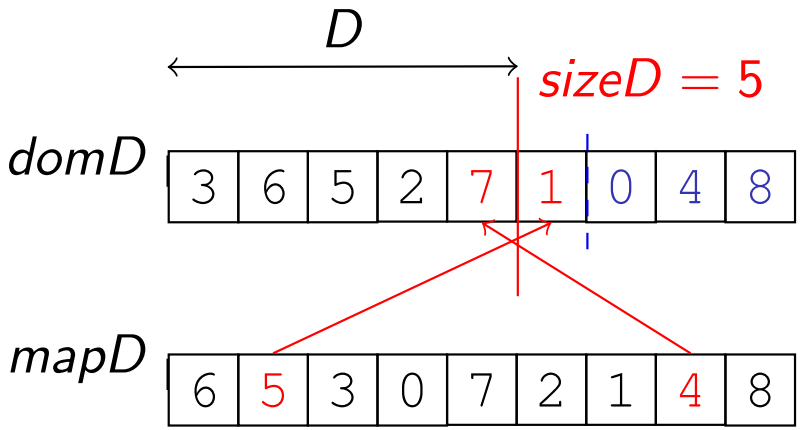
# Ensembles creux

## Opérations considérées

- $v \in D$ ? $\longrightarrow$ $mapD[v] < sizeD$ ? $\longrightarrow$ en temps constant

- Retirer $v$ de $D$ $\longrightarrow$ on échange $v$ avec le dernier élément dans $domD$, on décrémente $sizeD$ et on met à jour $mapD$ $\longrightarrow$ en temps constant



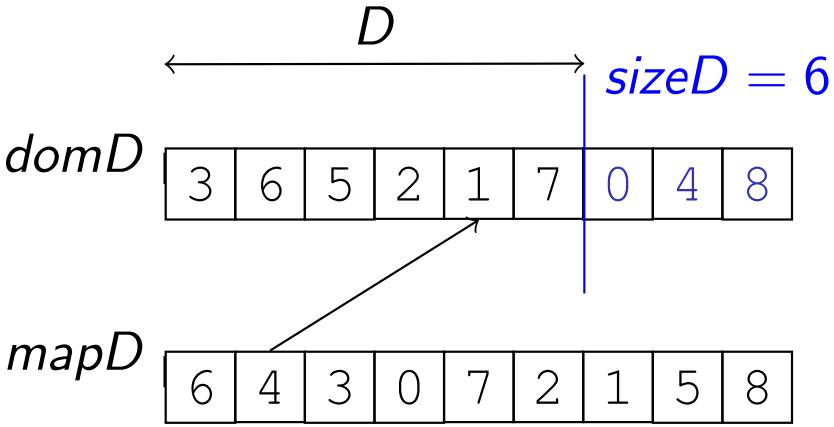



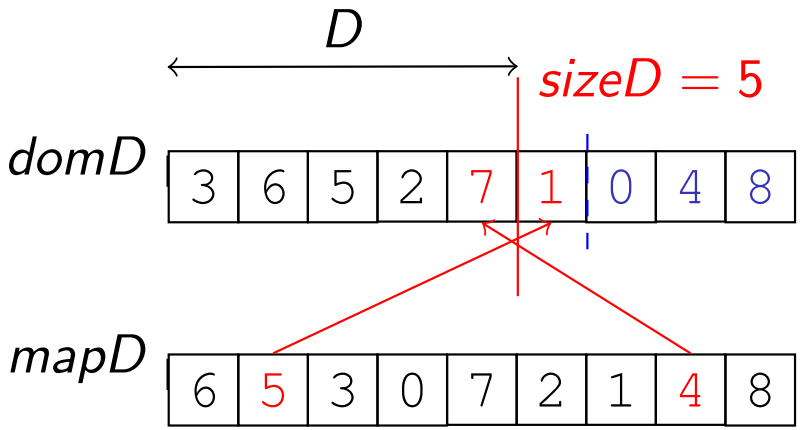
# Ensembles creux

## Opérations considérées

- $v \in D$ ? $\longrightarrow$ $mapD[v] < sizeD$ ? $\longrightarrow$ en temps constant

- Retirer $v$ de $D$ $\longrightarrow$ on échange $v$ avec le dernier élément dans $domD$, on décrémente $sizeD$ et on met à jour $mapD$ $\longrightarrow$ en temps constant



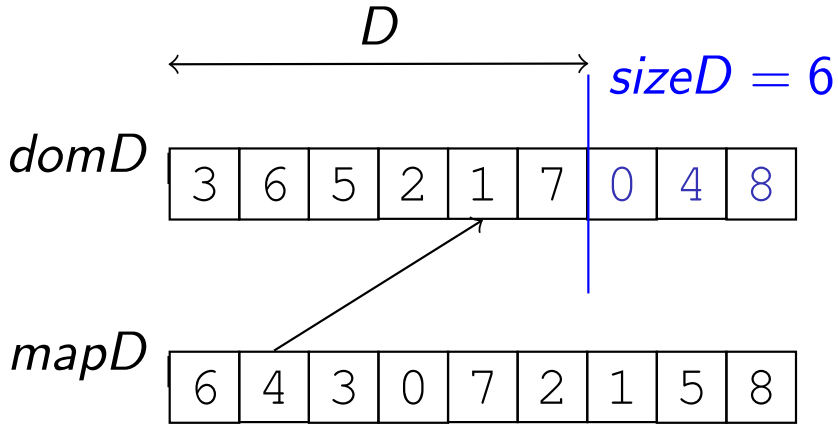

Sous-tableau de $domD$ en bleu identique après un retrait $(P_3)$

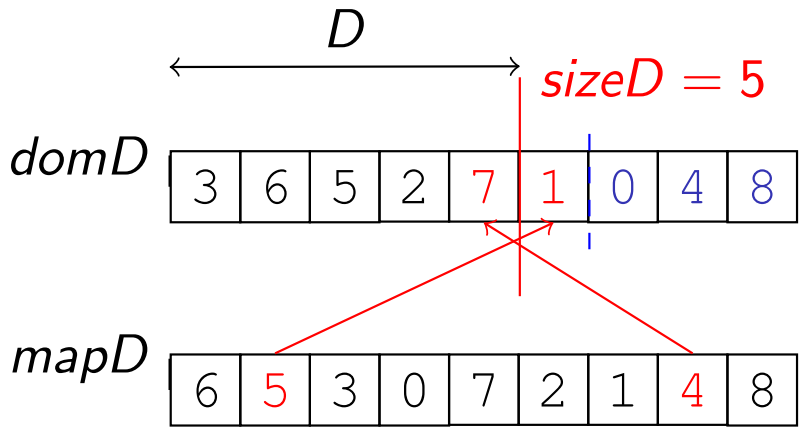
# Ensembles creux

## Opérations considérées

- $v \in D$ ? $\longrightarrow$ $mapD[v] < sizeD$ ? $\longrightarrow$ en temps constant

- Retirer $v$ de $D$ $\longrightarrow$ on échange $v$ avec le dernier élément dans $domD$, on décrémente $sizeD$ et on met à jour $mapD$ $\longrightarrow$ en temps constant




Sous-tableau de $domD$ en bleu identique après un retrait $(P_3)$

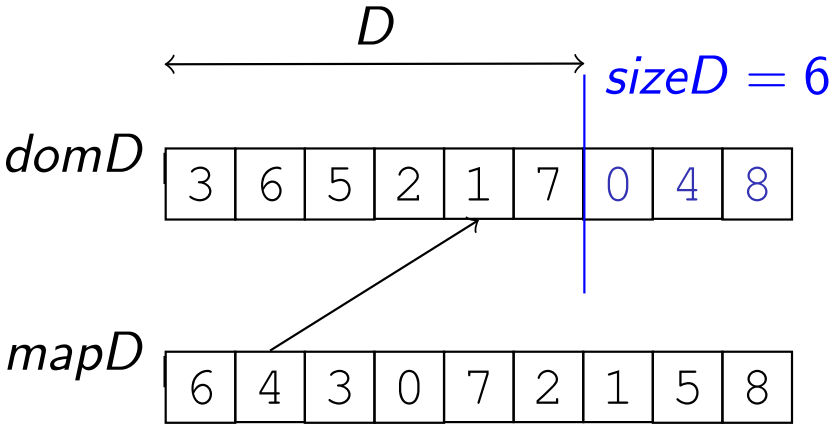
- $D \leftarrow \{v\}$ $\longrightarrow$ on échange $v$ avec le premier élément dans $domD$, on met $sizeD$ à 1 et on met à jour $mapD$ $\longrightarrow$ en temps constant

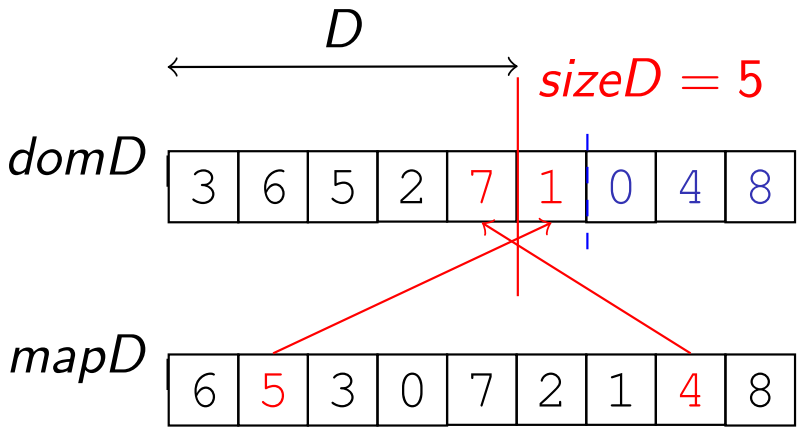
# Ensembles creux

## Opérations considérées

- $v \in D$ ? $\longrightarrow$ $mapD[v] < sizeD$ ? $\longrightarrow$ en temps constant

- Retirer $v$ de $D$ $\longrightarrow$ on échange $v$ avec le dernier élément dans $domD$, on décrémente $sizeD$ et on met à jour $mapD$ $\longrightarrow$ en temps constant




Sous-tableau de $domD$ en bleu identique après un retrait $(P_3)$

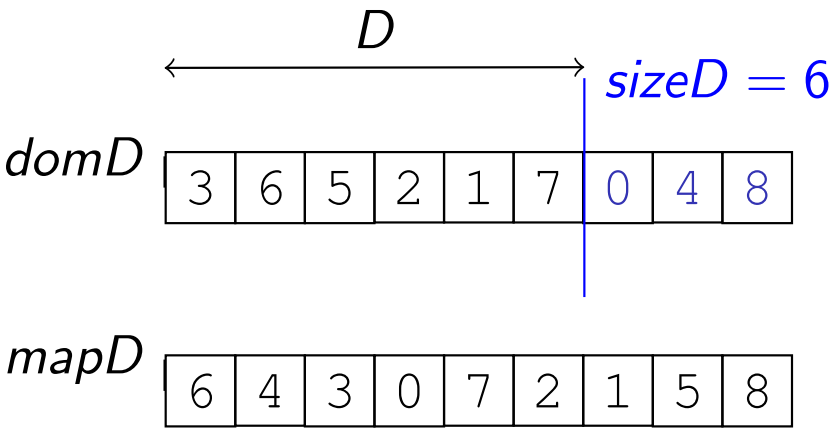
- $D \leftarrow \{v\}$ $\longrightarrow$ on échange $v$ avec le premier élément dans $domD$, on met $sizeD$ à 1 et on met à jour $mapD$ $\longrightarrow$ en temps constant

$\longrightarrow$ Propriétés $P_1$, $P_2$, $P_3$ vérifiées formellement

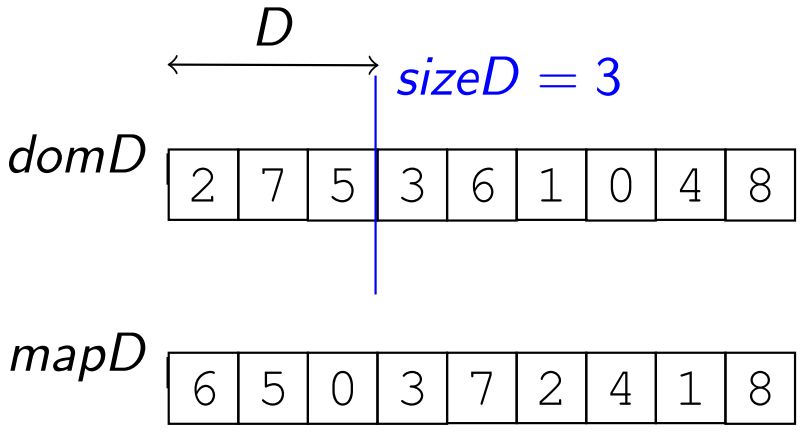
# Ensembles creux

## Structure de données *backtracable*

Le seul élément à sauvegarder et restaurer est *sizeD*

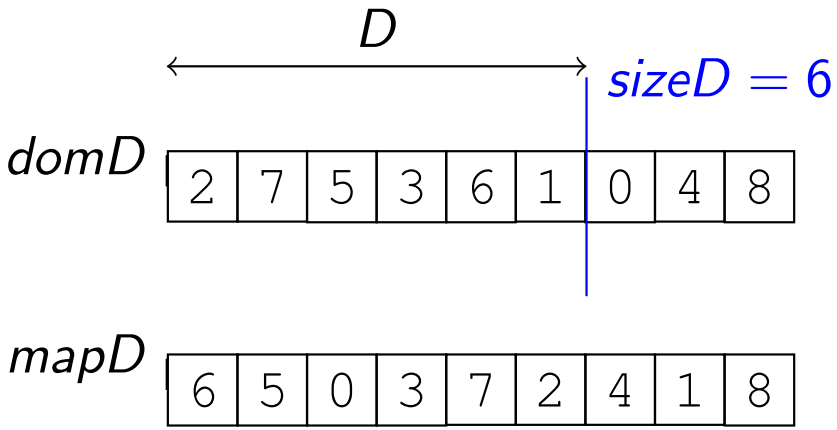



Save




Après retrait de 1, 6 et 3

Restore



# Un aperçu des modèles

## En EventB

**MACHINE** Domain
**VARIABLES** D
**INVARIANTS** $D \subseteq 0..N-1$

**Event** remove $\widehat{=}$
**any** v **where** $v \in D$ **then**
$D := D \setminus \{v\}$
**end**

**MACHINE** SparseSets_ref2
**REFINES** Domain
**VARIABLES** domD mapD sizeD
**INVARIANTS**
inv1 : $domD \in 0\,..\,N-1 \to 0\,..\,N-1$
inv2 : $mapD \in 0\,..\,N-1 \to 0\,..\,N-1$
inv3 : $sizeD \in 0\,..\,N$ $(P_2)$
inv4 : $domD\,;\,mapD = \mathrm{id}_{0..N-1}$
inv5 : $mapD\,;\,domD = \mathrm{id}_{0..N-1}$
inv6 : $domD[0\,..\,sizeD-1] = D$ $(P_1)$

**Event** remove $\widehat{=}$ **refines** remove
**any** v **where** $v \in 0\,..\,N-1 \wedge mapD(v) < sizeD$ **then**
$domD := domD \lhd\!\!\!- \ldots$
$mapD := mapD \lhd\!\!\!- \ldots$
$sizeD := sizeD - 1$
**end**

# Un aperçu des modèles

## En EventB

**MACHINE** Domain
**VARIABLES** D
**INVARIANTS** $D \subseteq 0..N-1$
**Event** remove $\widehat{=}$
  **any** v **where** $v \in D$ **then**
  $D := D \setminus \{v\}$
  **end**

**MACHINE** SparseSets_ref1
**REFINES** Domain
**VARIABLES** domD mapD sizeD
**INVARIANTS**
  inv1 : $domD \in 0\,..\,N-1 \to 0\,..\,N-1$
  inv2 : $mapD \in 0\,..\,N-1 \to 0\,..\,N-1$
  inv3 : $sizeD \in 0\,..\,N$ $(P_2)$
  inv4 : $domD\,;\,mapD = \mathrm{id}_{0..N-1}$
  inv5 : $mapD\,;\,domD = \mathrm{id}_{0..N-1}$
  inv6 : $domD[0\,..\,sizeD-1] = D$ $(P_1)$
**Event** remove $\widehat{=}$ **refines** remove
  **any** v **where** $v \in 0\,..\,N-1 \wedge mapD(v) < sizeD$ **then**
  $mapD, domD, sizeD :| \ldots \wedge$
  $(sizeD\,..\,N-1) \lhd domD' = (sizeD\,..\,N-1) \lhd DomD$ $(P_3)$
  **end**

**MACHINE** SparseSets_ref2
**REFINES** SparseSets_ref1
**VARIABLES** domD mapD sizeD
**Event** remove $\widehat{=}$ **refines** remove
  **any** v **where** $v \in 0\,..\,N-1 \wedge mapD(v) < sizeD$ **then**
  $domD := domD \mathrel{⩤} \ldots$
  $mapD := mapD \mathrel{⩤} \ldots$
  $sizeD := sizeD - 1$
  **end**

# Un aperçu des modèles

## En {log}

```
variables([D,DomD,MapD,SizeD]).

invariant(inv1a).
inv12(N,DomD) :- pfun(DomD) & N1 is N - 1 & dom(DomD,int(0,N1)).

operation(remove).
remove(N,SizeD,MapD,DomD,V,SizeD_,MapD_,DomD_) :-
  M is N - 1 & V in int(0,M) & 0 < SizeD & S is SizeD - 1 &
  MapD = [V,Y1],[Y2,Y4] / MapD1 & disj([V,Y1],[Y2,Y4],MapD1) & Y1 < SizeD
  DomD = [S,Y2],[Y1,Y5] / DomD1 & disj([S,Y2],[Y1,Y5],DomD1) &
  DomD_ = [S,V],[Y1,Y2] / DomD1 & MapD_ = [V,S],[Y2,Y1] / MapD1 &
  SizeD_ = S .

inv6(D,DomD,SizeD) :-
  S is SizeD - 1 & foreach([X,Y] in DomD, X in int(0,S) implies Y in D) &
  foreach(X in D, exists([A,B] in DomD, A in int(0,S) & B = X)).

theorem(remove_P1).
remove_P1(N,SizeD,MapD,DomD,V,SizeD_,MapD_,DomD_) :- ...

theorem(remove_P3).
remove_P3(N,SizeD,MapD,DomD,V,SizeD_,MapD_,DomD_) :- ....
```

# Un aperçu des modèles

## En WhyML

```
type data = {n: int; mutable domD, mapD : array int; mutable sizeD: int;}
invariant ran domD n && ran mapD n && 0 <= sizeD <= n &&
   forall v,i.  (0<=i<n &&  0<=v<n) -> (domD[i]=v <-> mapD[v]=i)      P2

type sparse = {mutable dt: data; mutable ghost setD: fset int;}
invariant subset setD (interval 0 dt.n) &&
   forall x: int.(exists i:int. 0 <= i < dt.sizeD && x = dt.domD[i])
                 <-> mem x setD                                       P1

let remove_sparse (v : int) (a : sparse)
requires {0<=v<a.dt.n && a.dt.mapD[v] < a.dt.sizeD}
ensures {a.setD == remove v (old a.setD) }
ensures {same_end a.dt.domD (old a.dt.domD) (old a.dt.sizeD) a.dt.n}  P3
=
  swap_data a.dt a.dt.mapD[v] (a.dt.sizeD - 1);
  a.dt.sizeD <- a.dt.sizeD - 1;
  a.setD <- remove v a.setD
```

# Effort de vérification

| Outil | VC | Automatique | Interactif | Temps |
|---|---|---|---|---|
| Rodin [1] | 46 | 34 | 12 | |
| {log} | 38 (6) | 38 | 0 | 7 mn |
| Why3 [2] | 53 | 51 | 2 | 9 s |

1. pp, pr, VeriT
2. CVC4, Z3

# Conclusion et perspectives

## Bilan

- Vérification formelle de trois opérations sur les ensembles creux dans 3 formalismes
- Focus sur les opérations et propriétés principales requises pour un solveur de contraintes
- Code disponible (`https://gitlab.com/cdubois/SparseSets`)

# Conclusion et perspectives

## Perspectives

- Implémentation et vérification formelle des autres opérations (union, intersection, différence, iter, forall, exists, copy, subset, cardinal) $\longrightarrow$ presque finie en WhyML/Why3
- Extraction de code depuis les modèles et comparaison des performances
- Intégration dans un solveur CP(FD) (backtracking avec sauvegarde et restauration, [Scherer 2023, Minicp 2021] en cours)
- Formalisation en Coq avec extraction de structures mutables [Sakaguchi 2020]