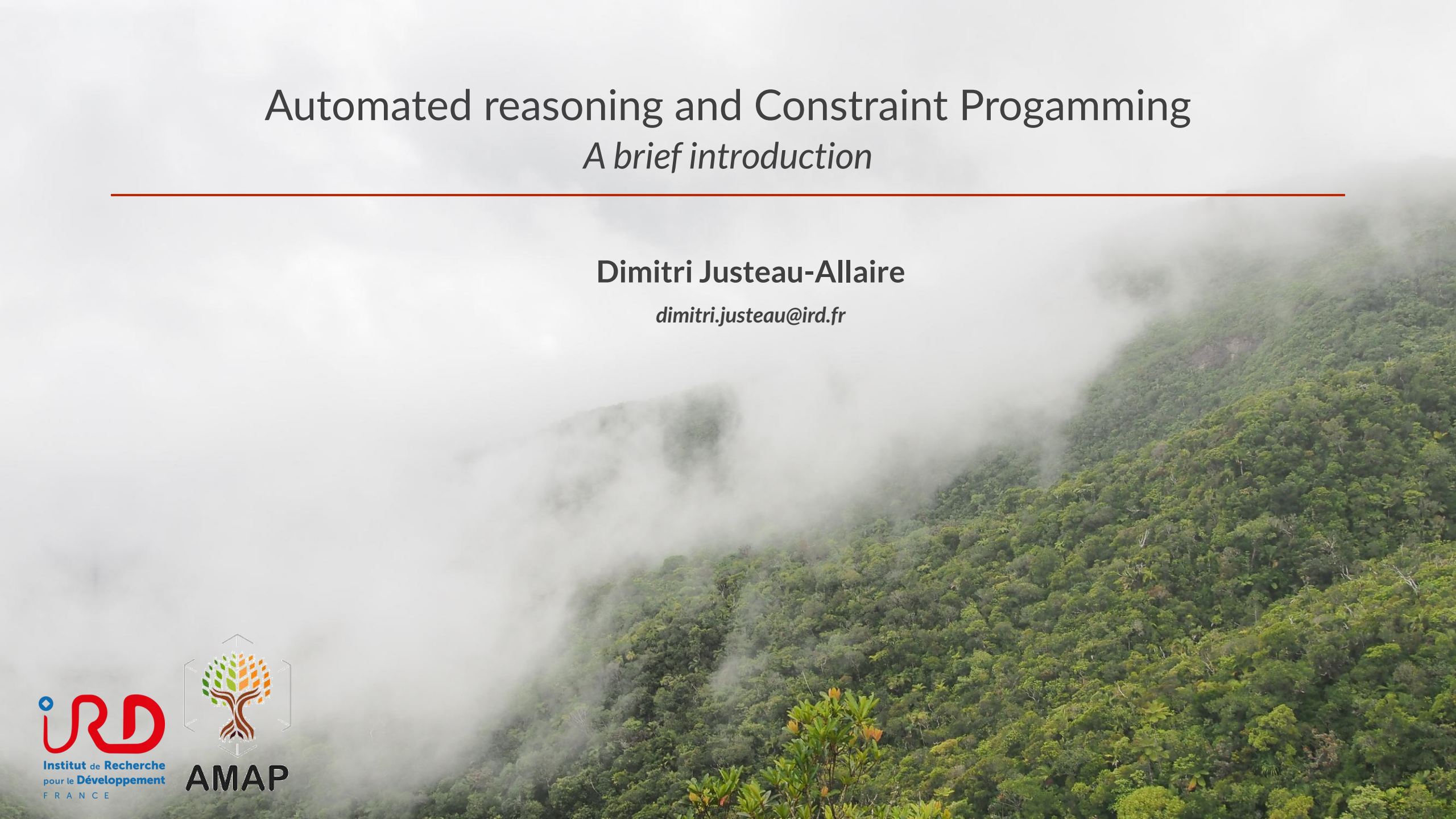
Automated reasoning and Constraint Progamming

A brief introduction

Dimitri Justeau-Allaire

dimitri.justeau@ird.fr

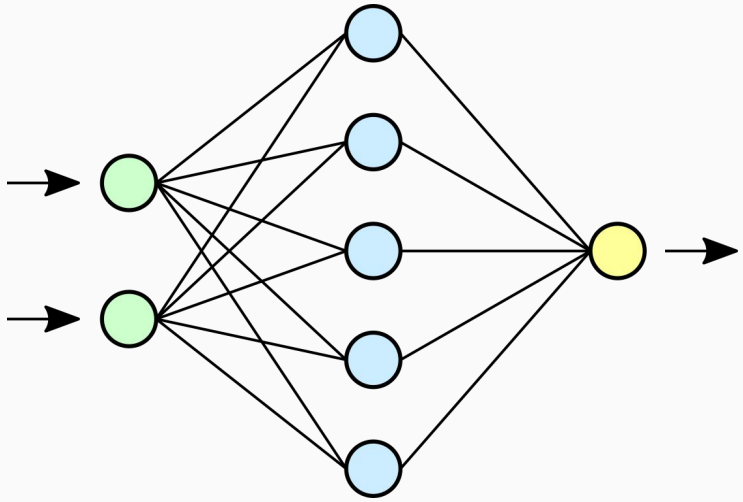


Institut de Recherche pour le Développement
FRANCE

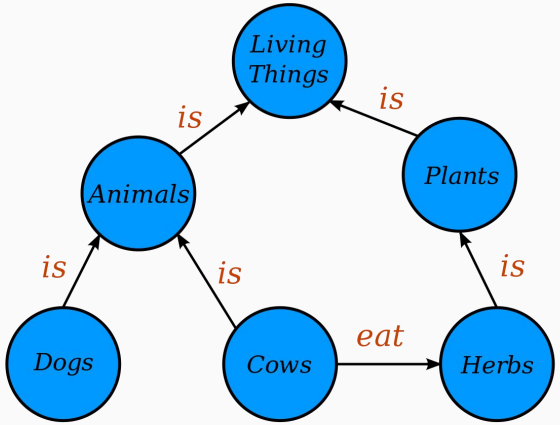
AMAP

Three main «domains» of artificial intelligence

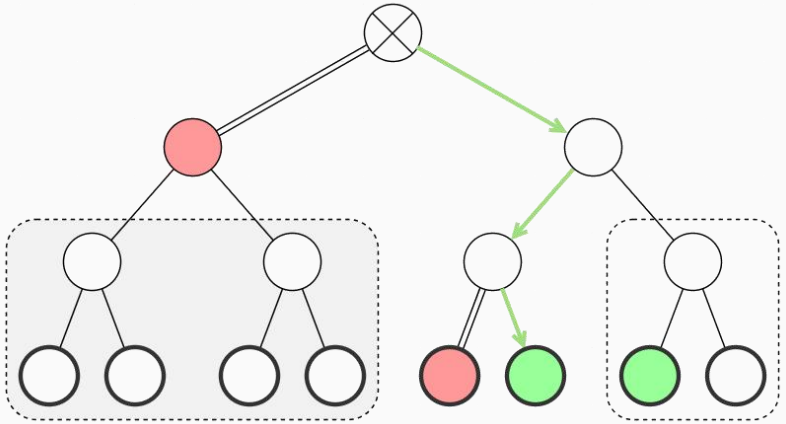
Automated learning



Knowledge representation

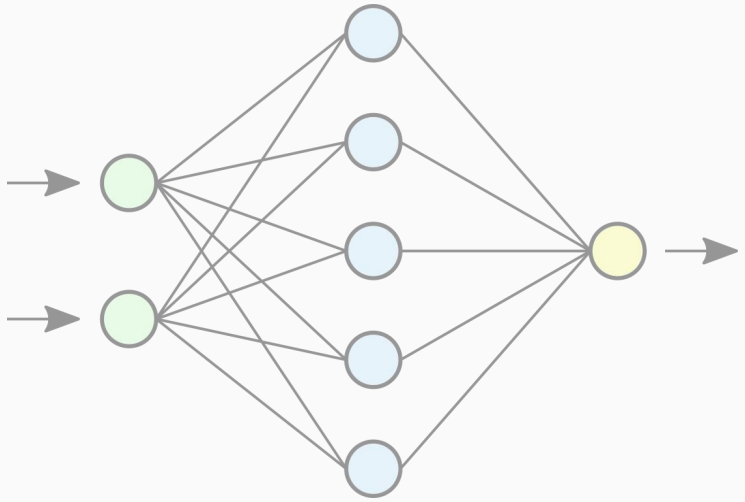


Automated reasoning

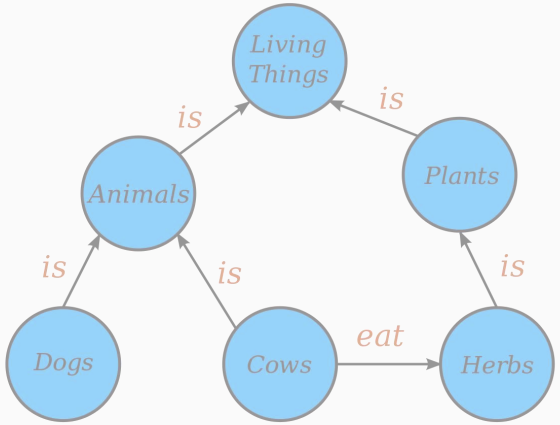


Three main «domains» of artificial intelligence

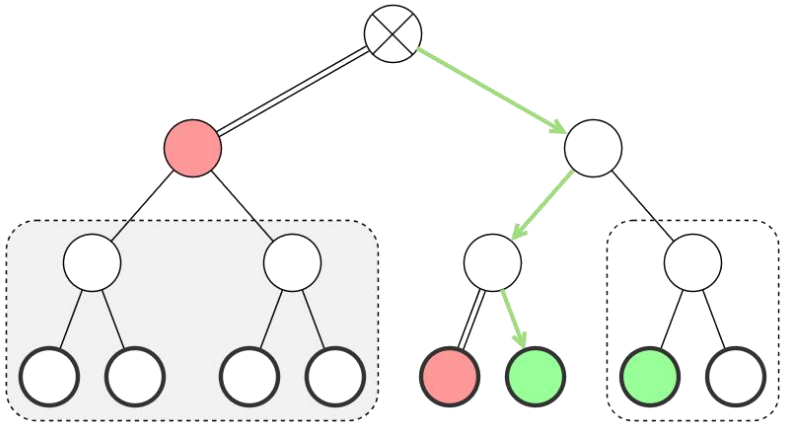
Automated learning



Knowledge representation



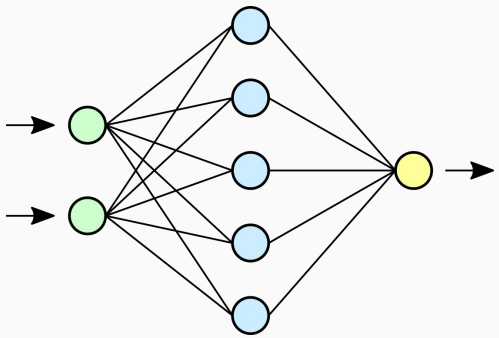
Automated reasoning

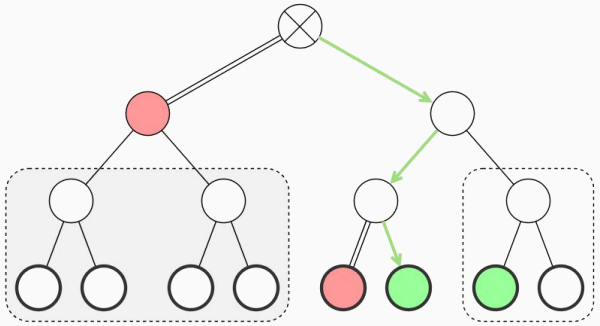


e.g. SAT, MILP, CP, expert systems, inference

Automated learning

Automated reasoning





Automated learning

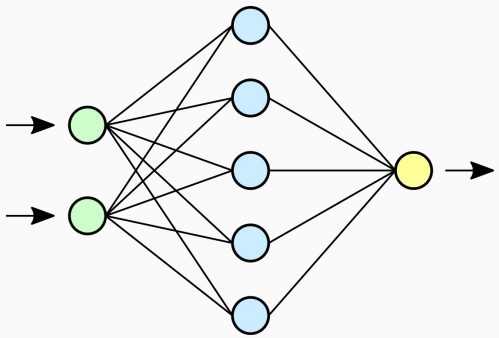
- Based on **data**

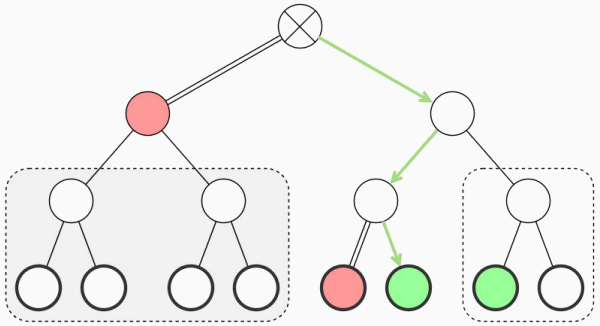
Automated reasoning

- Based on **models**

e.g. Equations, rules, constraints, formulas

$$(b_1 \wedge b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge b_4 \wedge b_2)$$





Automated learning

- Based on **data**
- Generic methods to perfom **complex tasks**

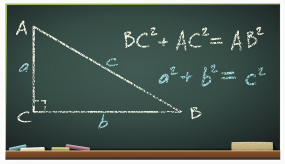
Automated reasoning

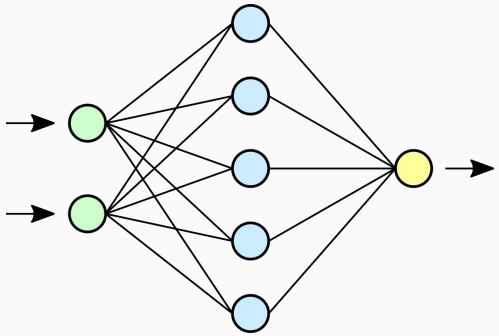
- Based on **models**
- Generic methods to solve **complex problems**

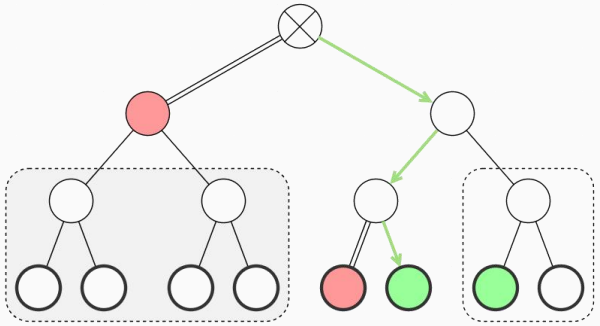
e.g. Equations, rules, constraints, formulas

$$(b_1 \wedge b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge b_4 \wedge b_2)$$

e.g. Prove the Pythagorean theorem

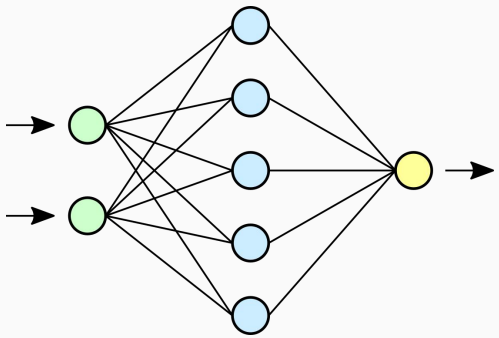






Automated learning

- Based on **data**
- Generic methods to perfom **complex tasks**
- Concerned about **accuracy**



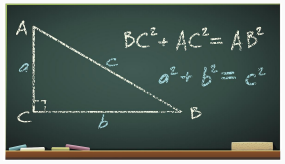
Automated reasoning

- Based on **models**
- Generic methods to solve **complex problems**
- Concerned about **guarantees**

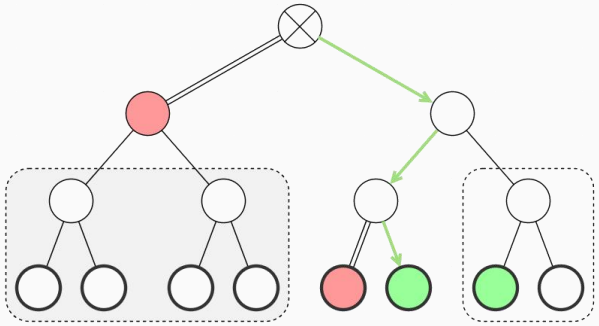
e.g. Equations, rules, constraints, formulas

$$(b_1 \wedge b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge b_4 \wedge b_2)$$

e.g. Prove the Pythagorean theorem

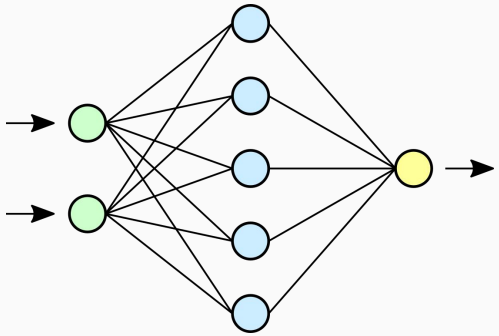


e.g. Guarantee that the flight control software computed optimal and safe trajectories



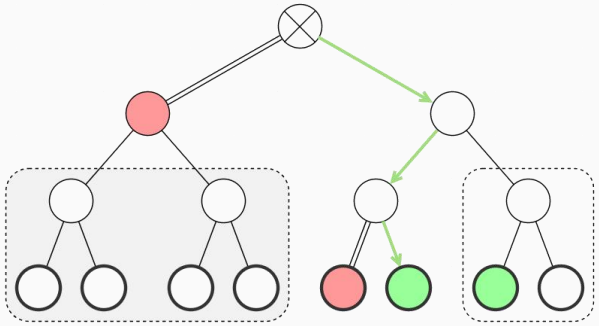
Automated learning

- Based on **data**
- Generic methods to perfom **complex tasks**
- Concerned about **accuracy**
- **Data** sensitive



Automated reasoning

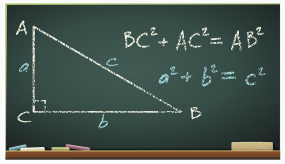
- Based on **models**
- Generic methods to solve **complex problems**
- Concerned about **guarantees**
- **Model** sensitive



e.g. Equations, rules, constraints, formulas

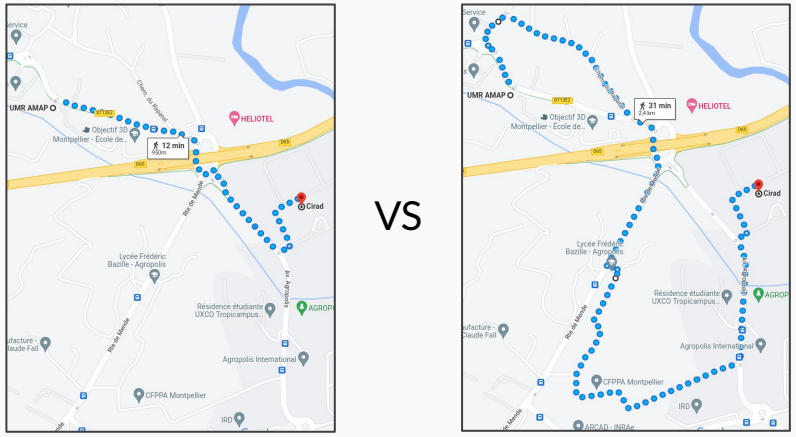
$$(b_1 \wedge b_2 \wedge \neg b_3) \vee (\neg b_1 \wedge b_4 \wedge b_2)$$

e.g. Prove the Pythagorean theorem



e.g. Guarantee that the flight control software computed optimal and safe trajectories

All roads lead to Rome, but some are faster than others...



- According to E. Freuder: 

  « Constraint Programming represents one of the closest approaches computer has yet made to the Holy Grail of programming: ***the user states the problem, the computer solves it****. »*

Introduction to **Constraint programming (CP)**

- According to E. Freuder: 

  « Constraint Programming represents one of the closest approaches computer has yet made to the Holy Grail of programming: ***the user states the problem, the computer solves it****. »*

- CP is paradigm for **modelling** and **solving** constraint statisfaction problem (**CSP**) and constrained optimization problems (**COP**).

- According to E. Freuder: 

  « Constraint Programming represents one of the closest approaches computer has yet made to the Holy Grail of programming: ***the user states the problem, the computer solves it****. »*

- CP is paradigm for **modelling** and **solving** constraint statisfaction problem (**CSP**) and constrained optimization problems (**COP**).

CSP: a triplet (X, D, C) where:

$X = \{x_1, \ldots, x_n\} \rightarrow$ variables
$D = \{D_1, \ldots, D_n\} \rightarrow$ domains
$C = \{C_1, \ldots, C_m\} \rightarrow$ constraints

- According to E. Freuder: 

  « Constraint Programming represents one of the closest approaches computer has yet made to the Holy Grail of programming: ***the user states the problem, the computer solves it****. »*

- CP is paradigm for **modelling** and **solving** constraint statisfaction problem (**CSP**) and constrained optimization problems (**COP**).

CSP: a triplet (X, D, C) where:

$X = \{x_1, \ldots, x_n\} \rightarrow$ variables
$D = \{D_1, \ldots, D_n\} \rightarrow$ domains
$C = \{C_1, \ldots, C_m\} \rightarrow$ constraints

COP: a CSP with:

$x_o \in X \rightarrow$ objective variable
A policy $\rightarrow$ *minimize* or *maximize*

- According to E. Freuder: 

  *« Constraint Programming represents one of the closest approaches computer has yet made to the Holy Grail of programming: **the user states the problem, the computer solves it**. »*

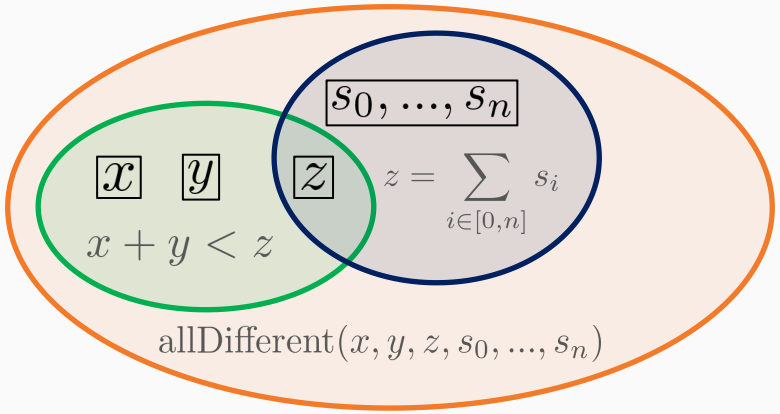
- CP is paradigm for **modelling** and **solving** constraint statisfaction problem (**CSP**) and constrained optimization problems (**COP**).

CSP: a triplet (X, D, C) where:

$X = \{x_1, \ldots, x_n\} \rightarrow$ variables
$D = \{D_1, \ldots, D_n\} \rightarrow$ domains
$C = \{C_1, \ldots, C_m\} \rightarrow$ constraints

COP: a CSP with:

$x_o \in X \rightarrow$ objective variable
A policy $\rightarrow$ *minimize* or *maximize*



- According to E. Freuder: 

  *« Constraint Programming represents one of the closest approaches computer has yet made to the Holy Grail of programming: **the user states the problem, the computer solves it**. »*

- CP is paradigm for **modelling** and **solving** constraint statisfaction problem (**CSP**) and constrained optimization problems (**COP**).

- Variables can be of several types: *Integers*, *Reals*, *Sets*, *Graphs*, *etc.*

- Constraints can be **extensional**, **arithmetic**, and **logical** (semantic)

  possible values are enumerated

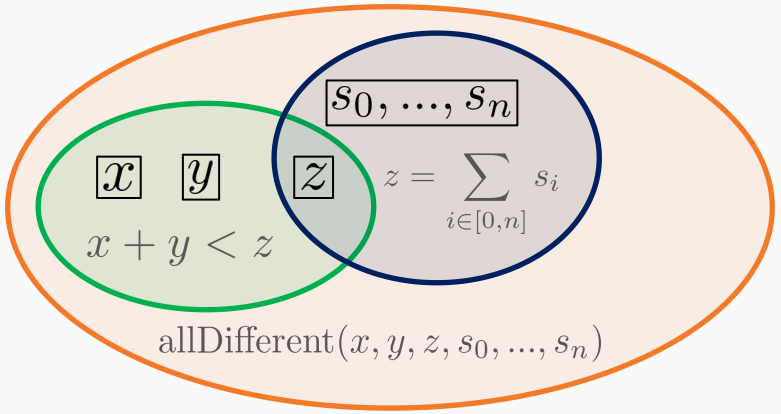
  e.g. AllDifferent, AtMost, Regular

CSP: a triplet (X, D, C) where:

$X = \{x_1, \ldots, x_n\} \rightarrow$ variables
$D = \{D_1, \ldots, D_n\} \rightarrow$ domains
$C = \{C_1, \ldots, C_m\} \rightarrow$ constraints

COP: a CSP with:

$x_o \in X \rightarrow$ objective variable
A policy $\rightarrow$ *minimize* or *maximize*



- According to E. Freuder: 

  *« Constraint Programming represents one of the closest approaches computer has yet made to the Holy Grail of programming: **the user states the problem, the computer solves it**. »*

- CP is paradigm for **modelling** and **solving** constraint statisfaction problem (**CSP**) and constrained optimization problems (**COP**).

- Variables can be of several types: *Integers*, *Reals*, *Sets*, *Graphs*, *etc.*

- Constraints can be **extensional**, **arithmetic**, and **logical** (semantic)

  possible values are enumerated

  e.g. AllDifferent, AtMost, Regular

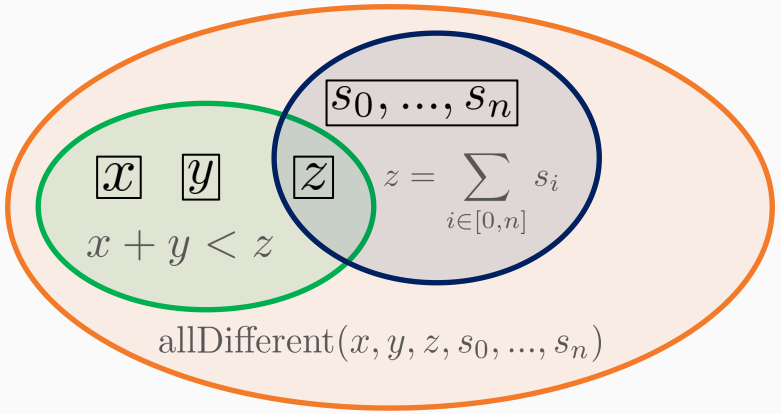
- Solving is **generic**, and based on **Filtering**, **Propagation**, and **Backtracking**

CSP: a triplet (X, D, C) where:

$X = \{x_1, \ldots, x_n\} \rightarrow$ variables
$D = \{D_1, \ldots, D_n\} \rightarrow$ domains
$C = \{C_1, \ldots, C_m\} \rightarrow$ constraints

COP: a CSP with:

$x_o \in X \rightarrow$ objective variable
A policy $\rightarrow$ *minimize* or *maximize*



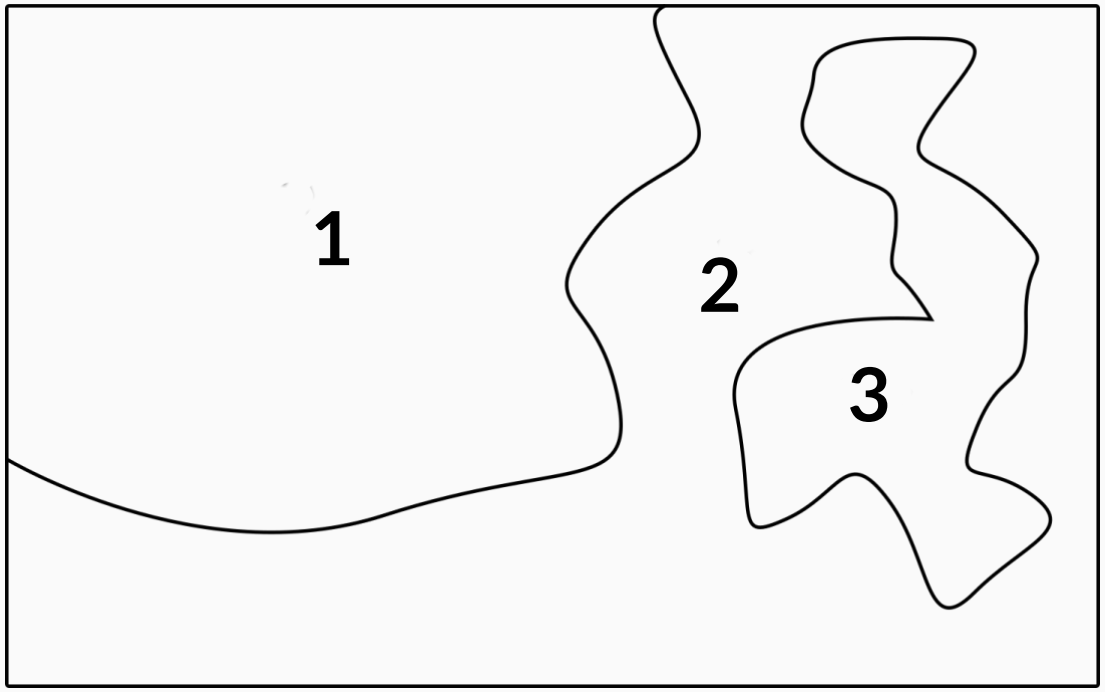
In summary, **CP** is:

- A generic approach for **modeling** and **solving constraint satisfaction** and **constrained optimization** problems
- Exact, expressive and extensible
- Support non-linear constraints
- A CP solver can interact with other systems
  - > e.g. Machine learning, SAT
- In practice:
  - > Modeling is ofter critical for performances
  - > When problems are hard, fine tuning is often necessary
- Many solvers, many languages

Google OR-Tools

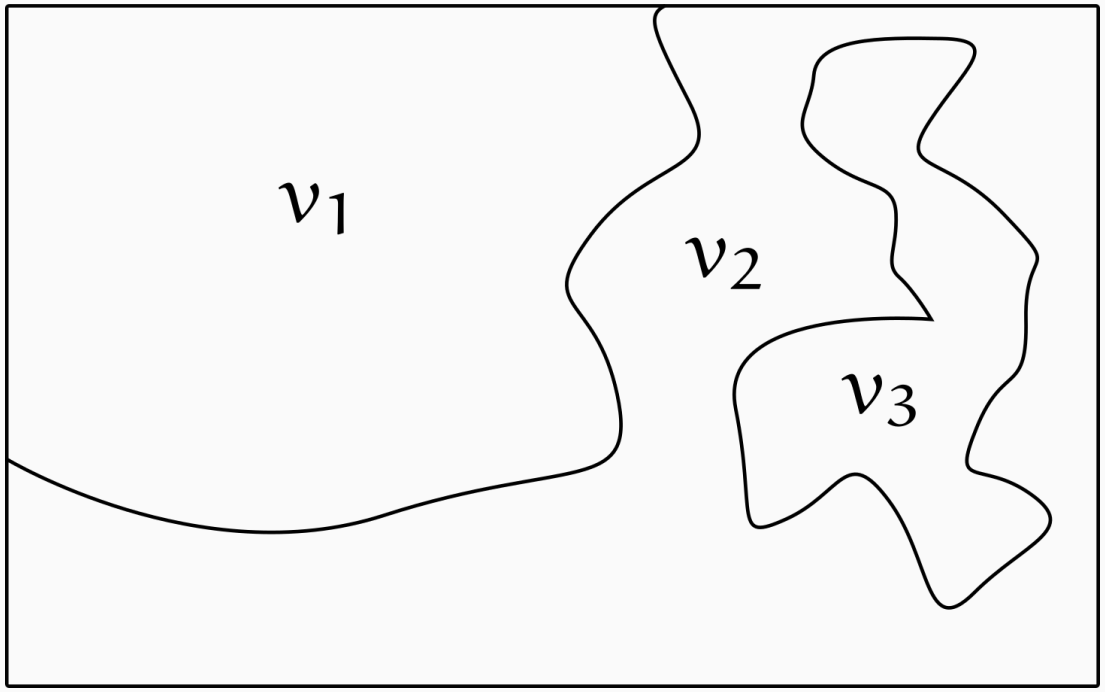
Modelling in CP: a basic example

Fill each region with a different color
1 can be blue, green or red
2 and 3 can be blue or green



Modelling in CP: a basic example

Fill each region with a different color
1 can be blue, green or red
2 and 3 can be blue or green



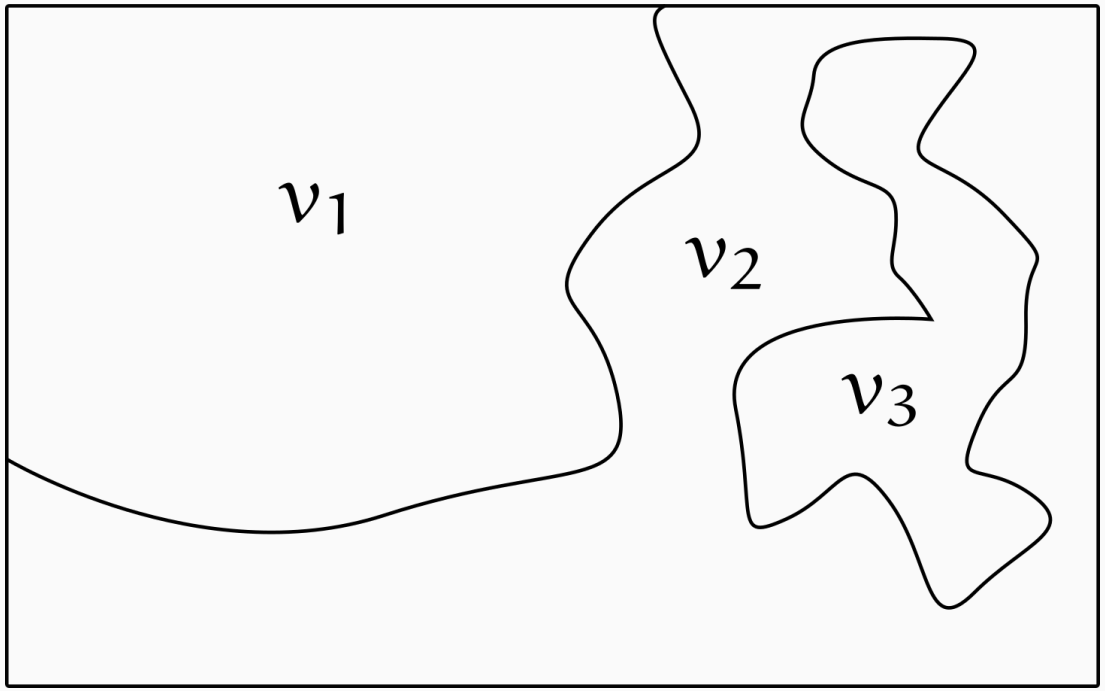
- For each region, one integer variable

$v_2 \in \{1, 2\}$

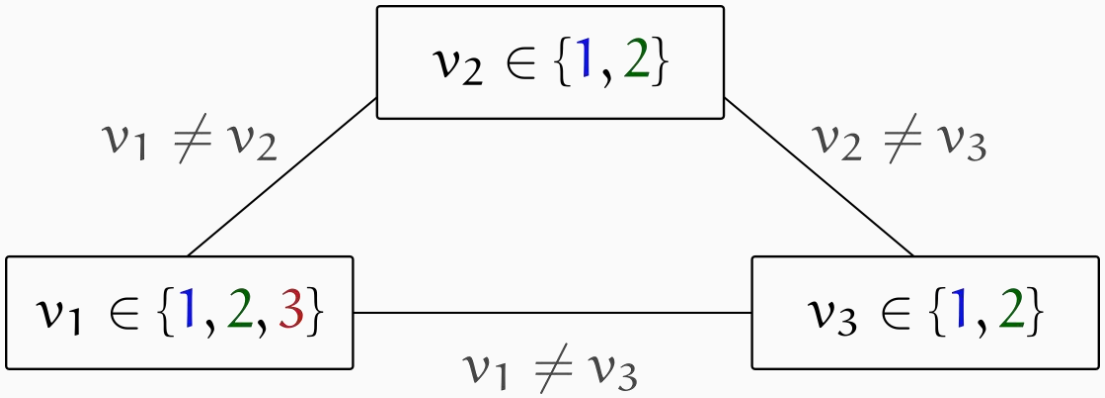
$v_1 \in \{1, 2, 3\}$

$v_3 \in \{1, 2\}$

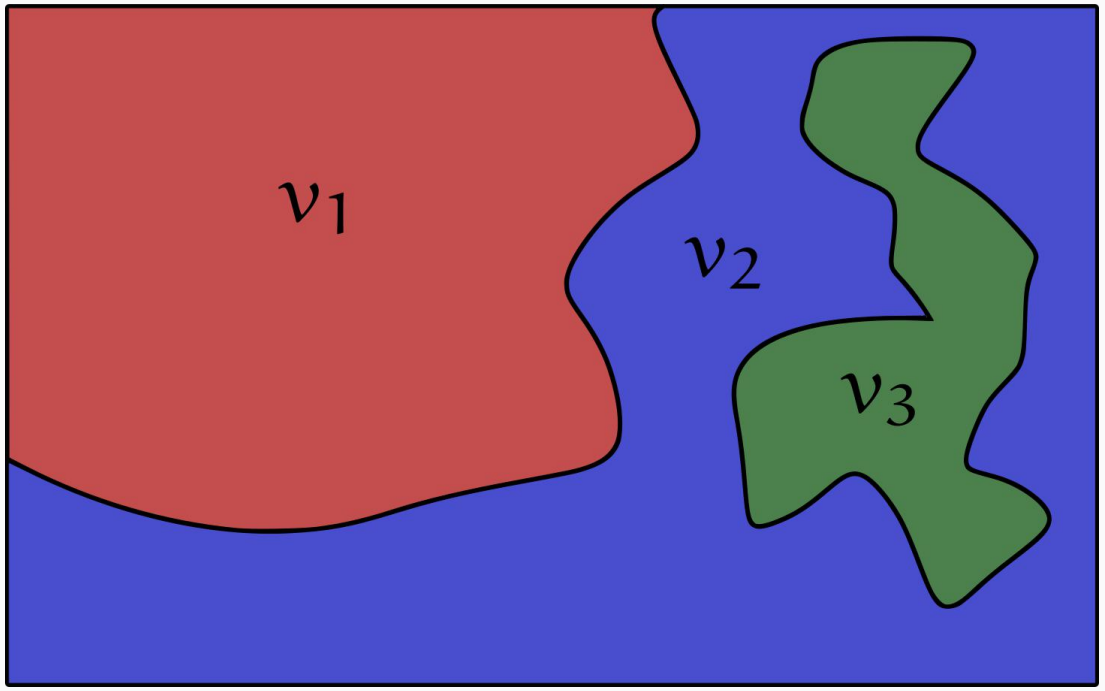
Fill each region with a different color
1 can be blue, green or red
2 and 3 can be blue or green



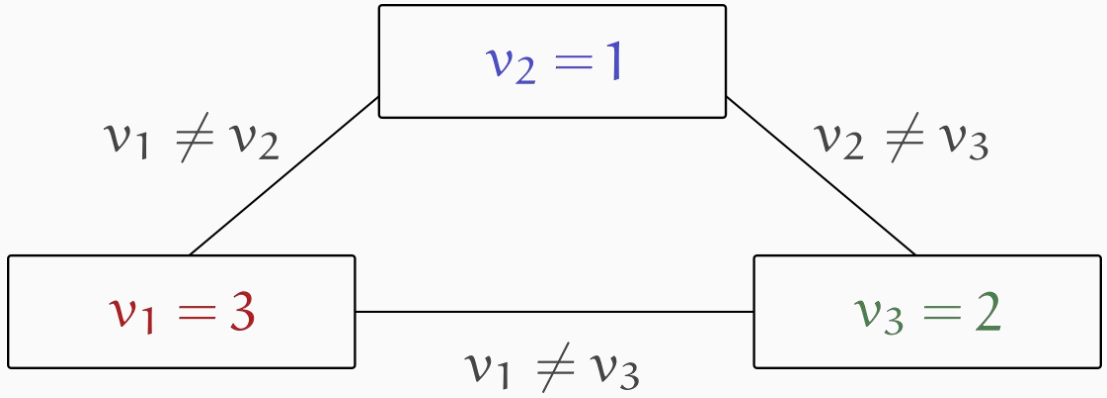
- For each region, one integer variable
- Inequality constraints



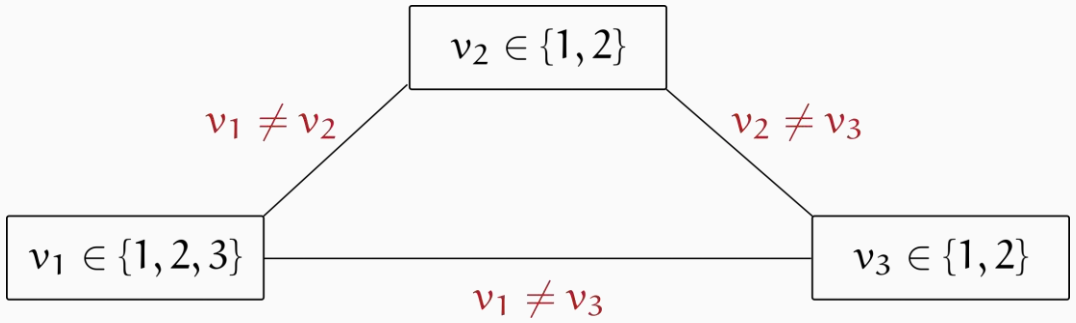
Fill each region with a different color
1 can be blue, green or red
2 and 3 can be blue or green



- For each region, one integer variable
- Inequality constraints
- An instantiation of the variables gives us a solution to the problem



What about the solving process ?



CP Model

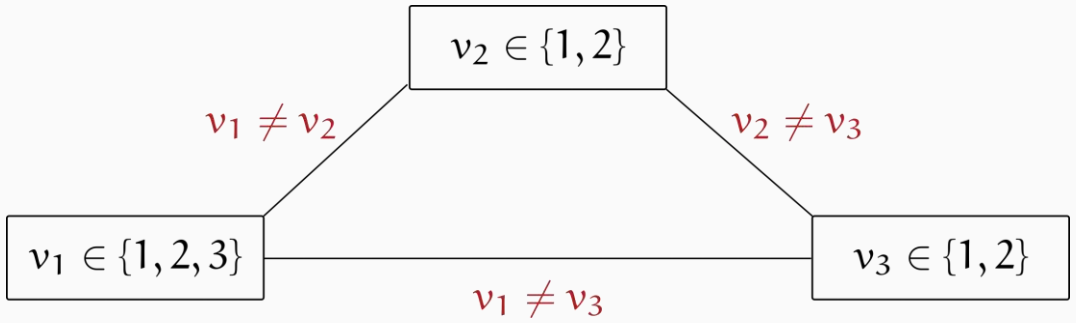
Step	v_1	v_2	v_3	Deductions
#0 *Initialize*	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$

Constraint propagation

$\mathcal{D}(v_1) = \{1,2,3\}$
$\mathcal{D}(v_2) = \{1,2\}$
$\mathcal{D}(v_3) = \{1,2\}$

Backtrackable search tree

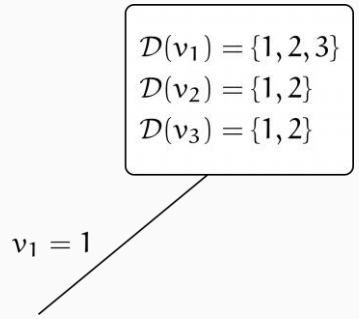
What about the solving process ?



CP Model

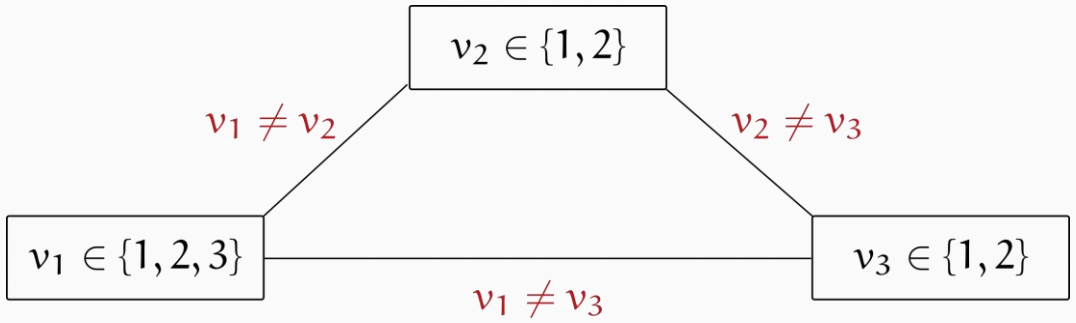
Step	v_1	v_2	v_3	Deductions
#0 *Initialize*	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#1 *Try* $v_1 = 1$	$\{1\}$	–	–	$v_2 \neq 1 \wedge v_3 \neq 1$

Constraint propagation



Backtrackable search tree

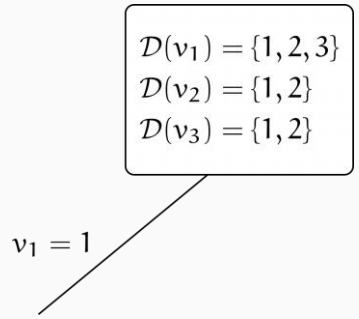
What about the solving process ?



CP Model

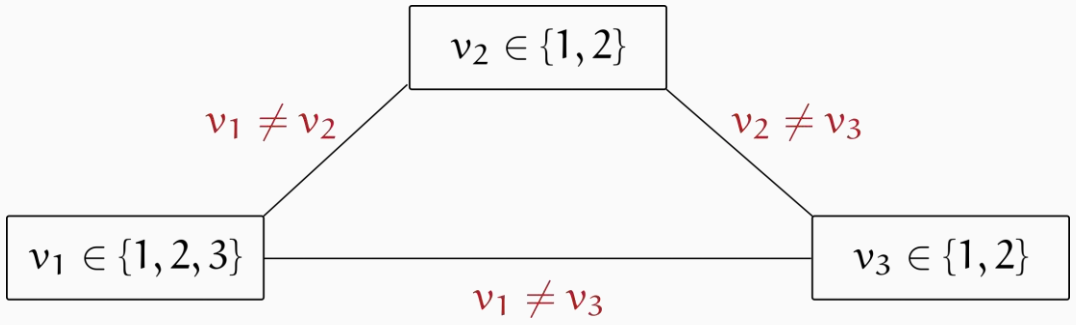
Step	v_1	v_2	v_3	Deductions
#0 *Initialize*	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#1 *Try* $v_1 = 1$	$\{1\}$	–	–	$v_2 \neq 1 \wedge v_3 \neq 1$
#2 *Remove* 1 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\{2\}$	$\{2\}$	$v_3 \neq 2 \wedge v_2 \neq 2$

Constraint propagation



Backtrackable search tree

What about the solving process ?

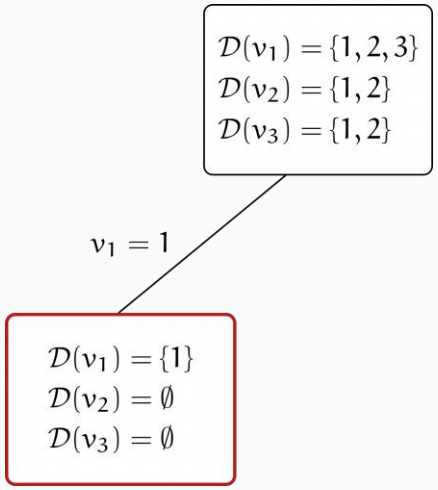


CP Model

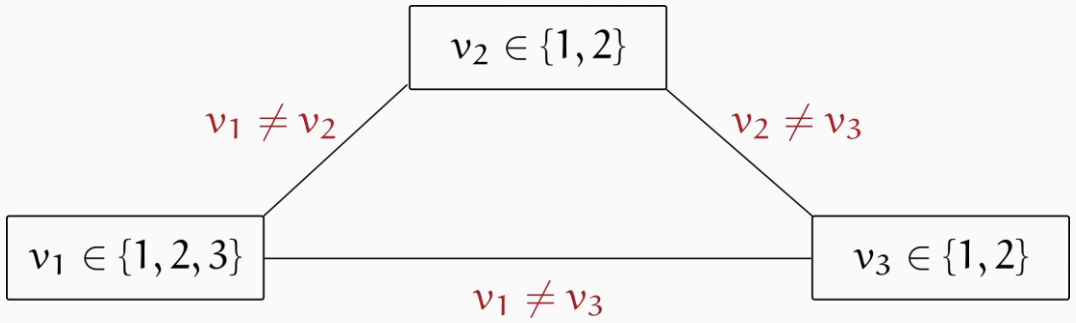
Step	v_1	v_2	v_3	Deductions
#0 *Initialize*	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#1 *Try* $v_1 = 1$	$\{1\}$	–	–	$v_2 \neq 1 \wedge v_3 \neq 1$
#2 *Remove* 1 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\{2\}$	$\{2\}$	$v_3 \neq 2 \wedge v_2 \neq 2$
#3 *Remove* 2 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\emptyset$	$\emptyset$	$v_1 \neq 1$

Constraint propagation

Backtrackable search tree



What about the solving process ?

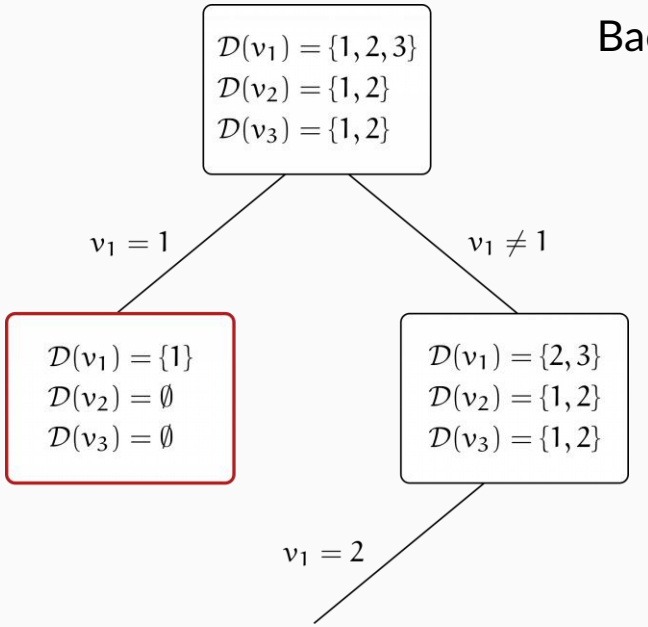


CP Model

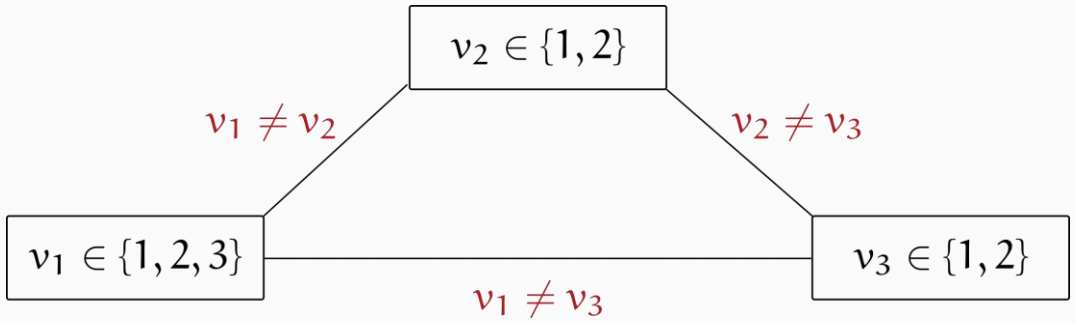
Step	v_1	v_2	v_3	Deductions
#0 *Initialize*	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#1 *Try* $v_1 = 1$	$\{1\}$	–	–	$v_2 \neq 1 \wedge v_3 \neq 1$
#2 *Remove* 1 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\{2\}$	$\{2\}$	$v_3 \neq 2 \wedge v_2 \neq 2$
#3 *Remove* 2 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\emptyset$	$\emptyset$	$v_1 \neq 1$
#4 *Backtrack to* #0 *and try* $v_1 = 2$	$\{2\}$	$\{1,2\}$	$\{1,2\}$	$v_2 \neq 2 \wedge v_3 \neq 2$

Constraint propagation

Backtrackable search tree



What about the solving process ?

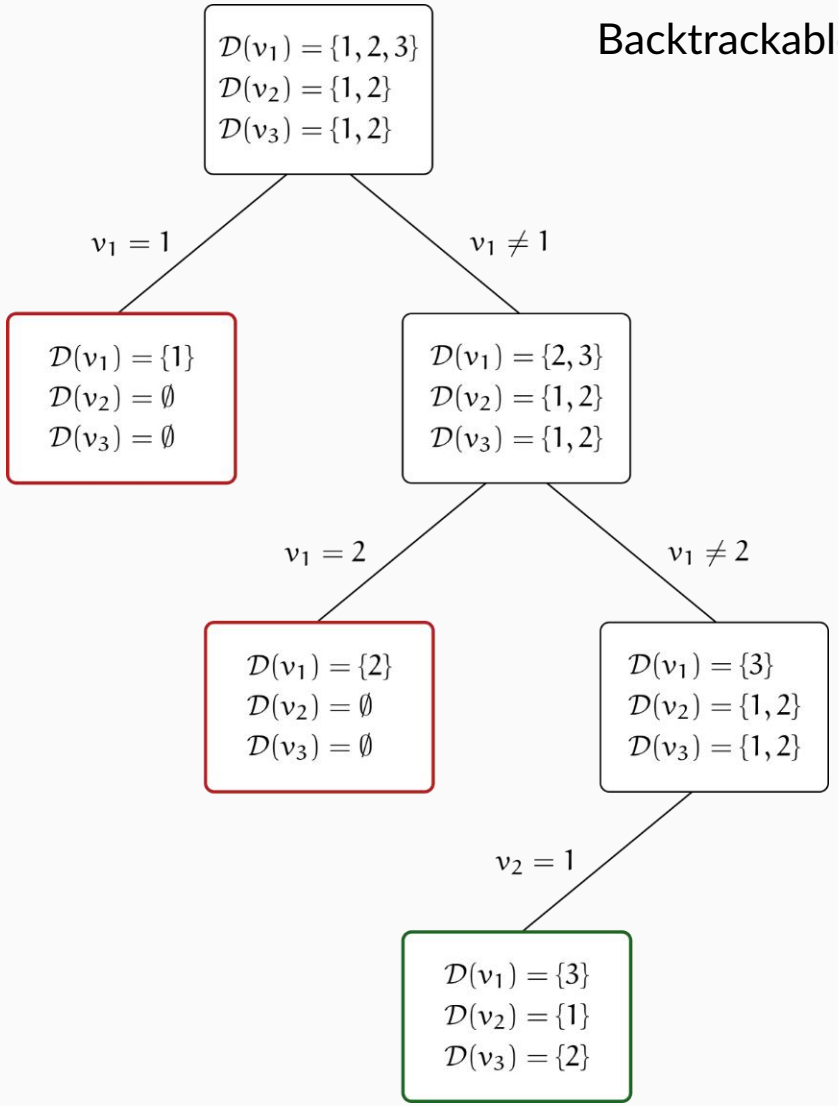


CP Model

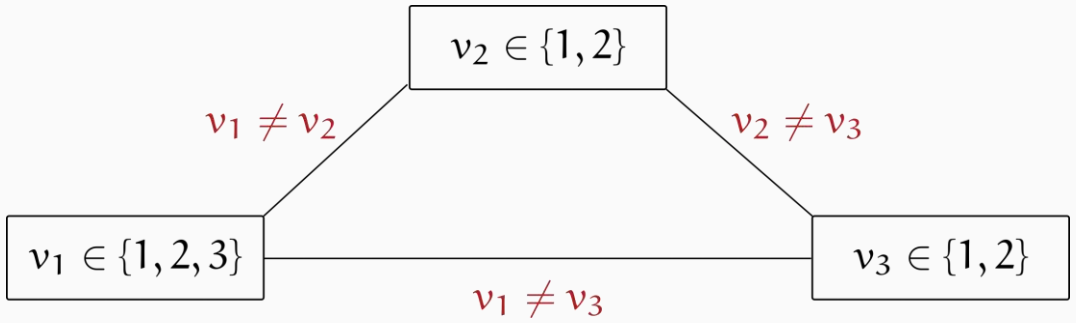
Step	v_1	v_2	v_3	Deductions
#0 *Initialize*	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#1 *Try* $v_1 = 1$	$\{1\}$	–	–	$v_2 \neq 1 \wedge v_3 \neq 1$
#2 *Remove* 1 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\{2\}$	$\{2\}$	$v_3 \neq 2 \wedge v_2 \neq 2$
#3 *Remove* 2 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\emptyset$	$\emptyset$	$v_1 \neq 1$
#4 *Backtrack to #0 and try* $v_1 = 2$	$\{2\}$	$\{1,2\}$	$\{1,2\}$	$v_2 \neq 2 \wedge v_3 \neq 2$
#5 *Remove* 2 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\{1\}$	$\{1\}$	$v_3 \neq 1 \wedge v_2 \neq 1$
#6 *Remove* 1 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\emptyset$	$\emptyset$	$v_1 \neq 2$
#7 *Backtrack to #0 and try* $v_1 = 3$	$\{3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#8 *Try* $v_2 = 1$	–	$\{1\}$	–	$v_3 \neq 1$
#9 *Remove* 1 *from* $\mathcal{D}(v_3)$	$\{3\}$	$\{1\}$	$\{2\}$	$(3,1,2)$ is a solution

Constraint propagation

Backtrackable search tree



What about the solving process ?

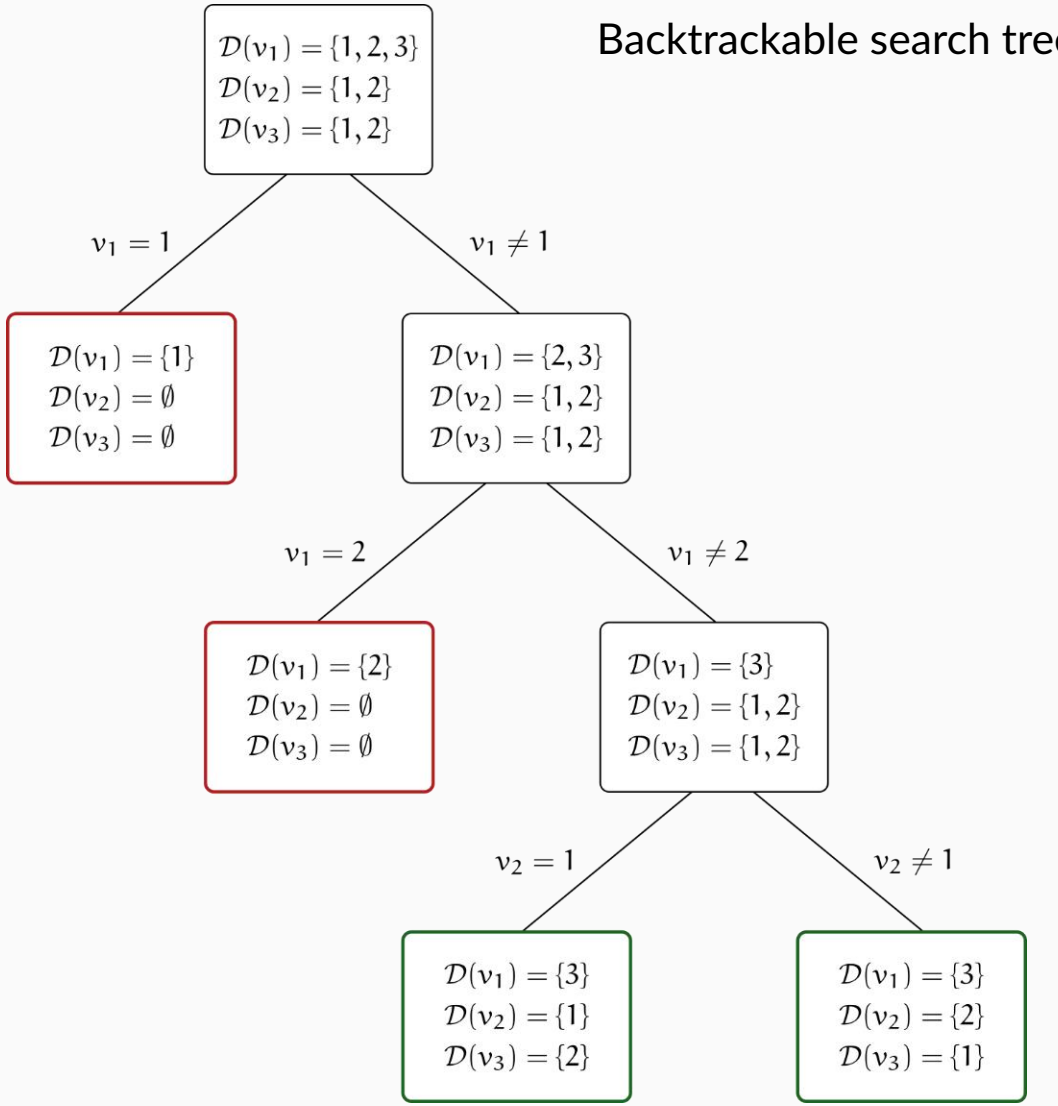


CP Model

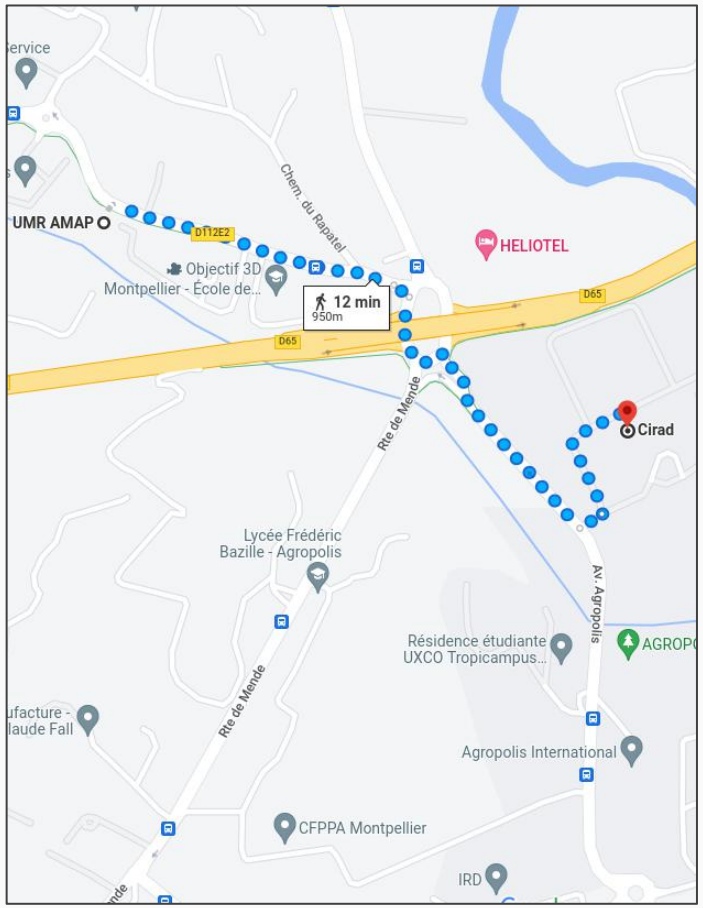
Step	v_1	v_2	v_3	Deductions
#0 *Initialize*	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#1 *Try* $v_1 = 1$	$\{1\}$	–	–	$v_2 \neq 1 \wedge v_3 \neq 1$
#2 *Remove* 1 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\{2\}$	$\{2\}$	$v_3 \neq 2 \wedge v_2 \neq 2$
#3 *Remove* 2 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\emptyset$	$\emptyset$	$v_1 \neq 1$
#4 *Backtrack to #0 and try* $v_1 = 2$	$\{2\}$	$\{1,2\}$	$\{1,2\}$	$v_2 \neq 2 \wedge v_3 \neq 2$
#5 *Remove* 2 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\{1\}$	$\{1\}$	$v_3 \neq 1 \wedge v_2 \neq 1$
#6 *Remove* 1 *from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\emptyset$	$\emptyset$	$v_1 \neq 2$
#7 *Backtrack to #0 and try* $v_1 = 3$	$\{3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#8 *Try* $v_2 = 1$	–	$\{1\}$	–	$v_3 \neq 1$
#9 *Remove* 1 *from* $\mathcal{D}(v_3)$	$\{3\}$	$\{1\}$	$\{2\}$	$(3,1,2)$ is a solution

Constraint propagation

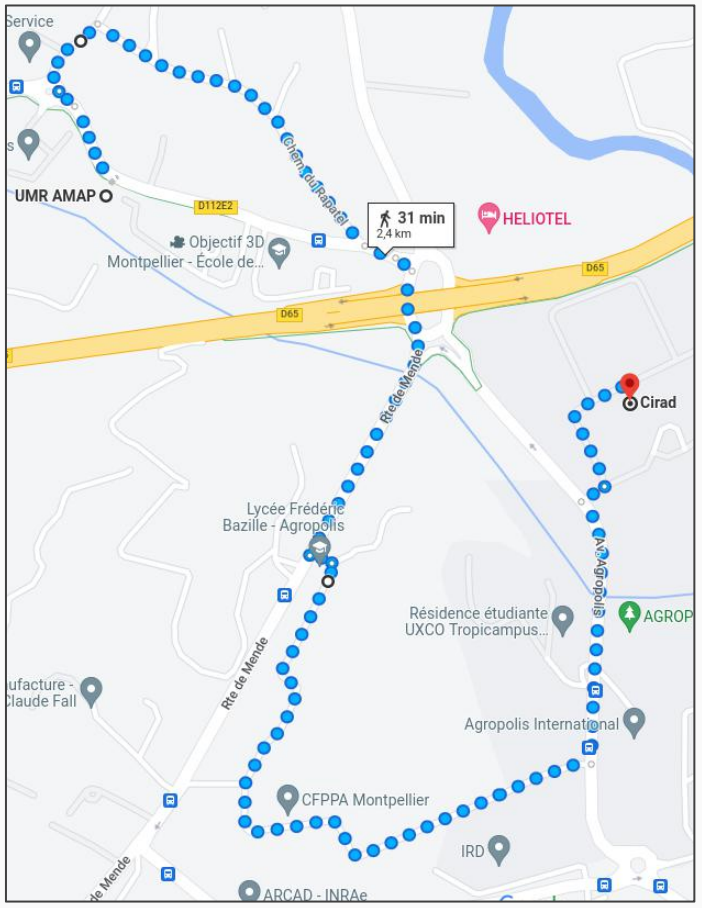
Backtrackable search tree



Do you remember ? *All roads lead to Rome, but some are faster than others...*

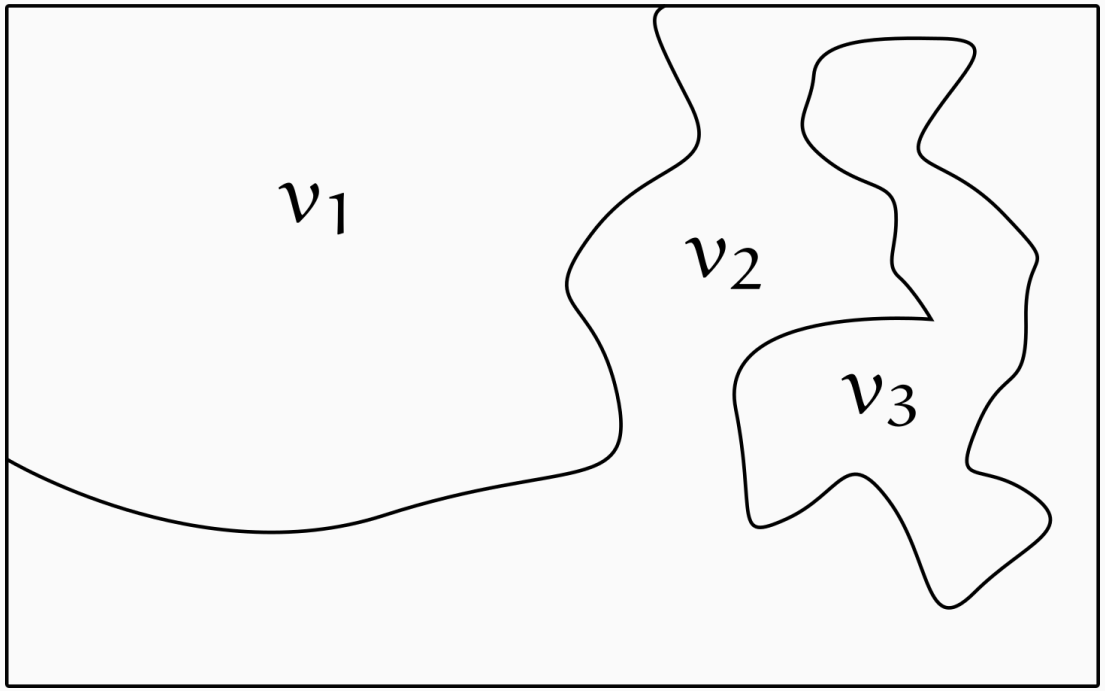


VS



A better model for our simple coloring problem ?

Fill each region with a different color
1 can be blue, green or red
2 and 3 can be blue or green

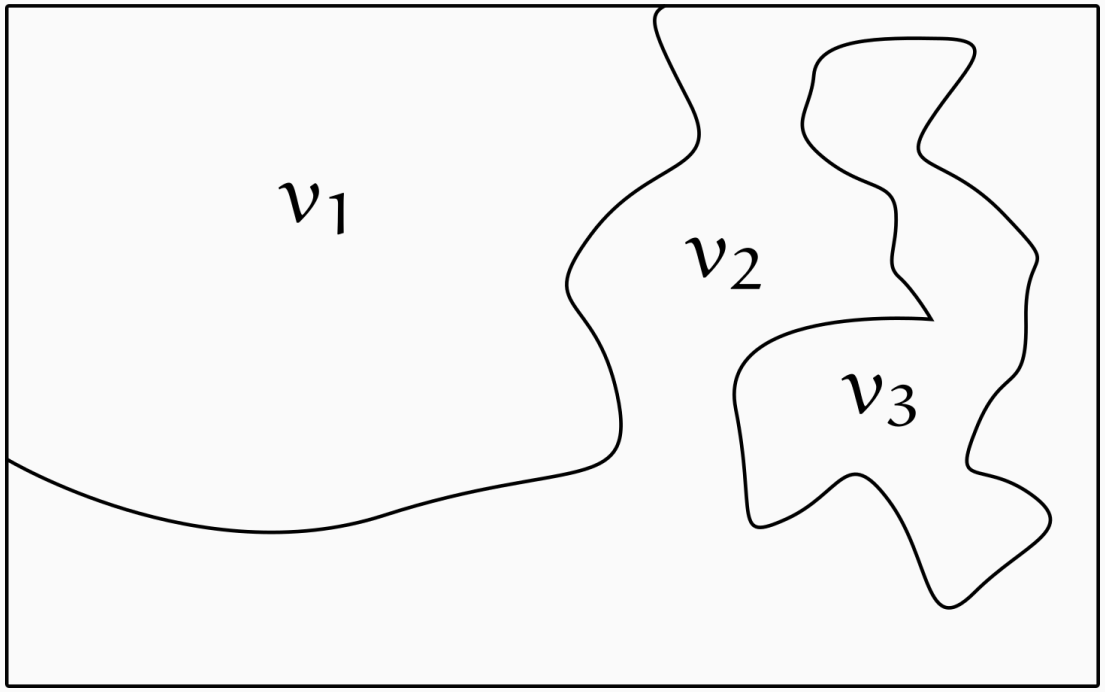


$v_2 \in \{1, 2\}$

$v_1 \in \{1, 2, 3\}$

$v_3 \in \{1, 2\}$

Fill each region with a different color
1 can be blue, green or red
2 and 3 can be blue or green



- **AllDifferent**, a global (logical) constraint

AllDifferent(v_1, v_2 , v_3)

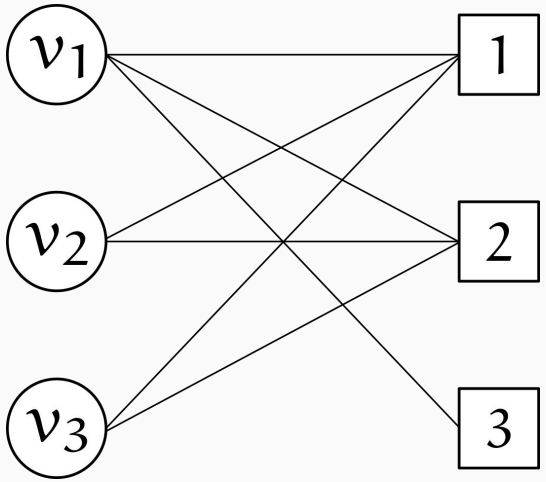
$v_2 \in \{1, 2\}$

$v_1 \in \{1, 2, 3\}$

$v_3 \in \{1, 2\}$

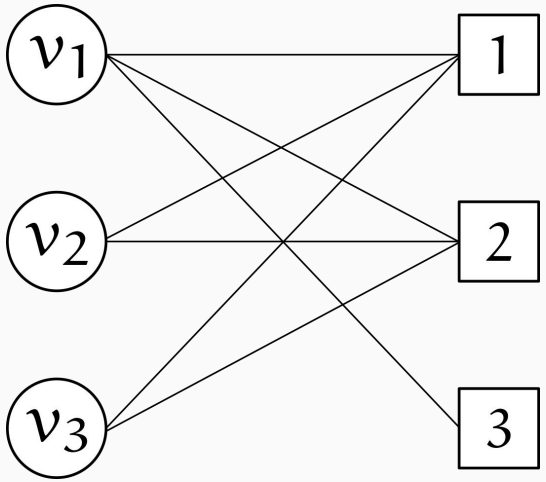
A global constraint example: **AllDifferent**

- Instead of applying pairwise binary inequality constraints,
- Variables and their possible values are seen as a **bipartite graph**



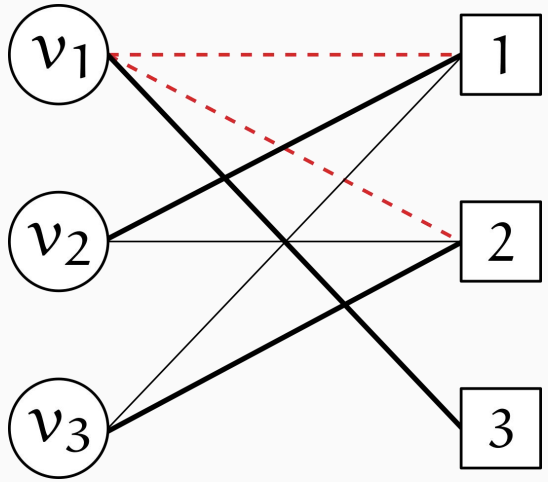
A global constraint example: **AllDifferent**

- Instead of applying pairwise binary inequality constraints,
- Variables and their possible values are seen as a **bipartite graph**
- A solution corresponds to a **maximum cardinality matching (MCM)**



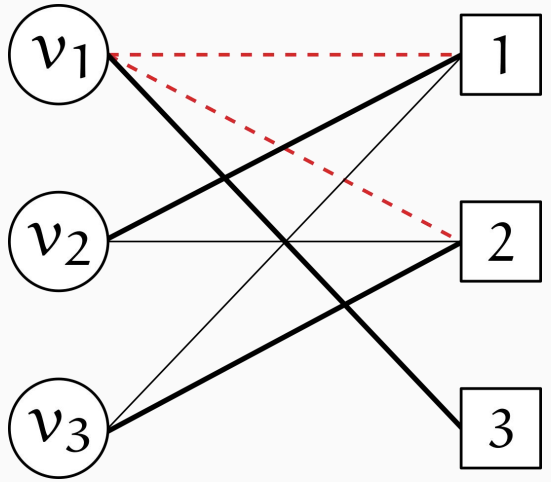
A global constraint example: **AllDifferent**

- Instead of applying pairwise binary inequality constraints,
- Variables and their possible values are seen as a **bipartite graph**
- A solution corresponds to a **maximum cardinality matching (MCM)**
- Filtering: remove edges **that cannot belong** to a MCM
  - > Algorithm based on the Hall's marriage theorem (linear time)



A global constraint example: **AllDifferent**

- Instead of applying pairwise binary inequality constraints,
- Variables and their possible values are seen as a **bipartite graph**
- A solution corresponds to a **maximum cardinality matching (MCM)**
- Filtering: remove edges **that cannot belong** to a MCM
  - > Algorithm based on the Hall's marriage theorem (linear time)



Step	v_1	v_2	v_3	Deductions
#0 *Initialize*	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#1 *Try* $v_1 = 1$	$\{1\}$	–	–	$v_2 \neq 1 \wedge v_3 \neq 1$
#2 *Remove 1 from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\{2\}$	$\{2\}$	$v_3 \neq 2 \wedge v_2 \neq 2$
#3 *Remove 2 from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\emptyset$	$\emptyset$	$v_1 \neq 1$
#4 *Backtrack to #0 and try* $v_1 = 2$	$\{2\}$	$\{1,2\}$	$\{1,2\}$	$v_2 \neq 2 \wedge v_3 \neq 2$
#5 *Remove 2 from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\{1\}$	$\{1\}$	$v_3 \neq 1 \wedge v_2 \neq 1$
#6 *Remove 1 from* $\mathcal{D}(v_2)$ *and* $\mathcal{D}(v_3)$	–	$\emptyset$	$\emptyset$	$v_1 \neq 2$
#7 *Backtrack to #0 and try* $v_1 = 3$	$\{3\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$
#8 *Try* $v_2 = 1$	–	$\{1\}$	–	$v_3 \neq 1$
#9 *Remove 1 from* $\mathcal{D}(v_3)$	$\{3\}$	$\{1\}$	$\{2\}$	$(3,1,2)$ is a solution

⇨

Step	v_1	v_2	v_3	Deductions
#0 *Initialize*	$\{1,2,3\}$	$\{1,2\}$	$\{1,2\}$	$v_1 \neq 1 \wedge v_1 \neq 2$
#1 *Remove 1 and 2 from* $\mathcal{D}(v_1)$	$\{3\}$	–	–	$\emptyset$
#2 *Try* $v_2 = 1$	–	$\{1\}$	–	$v_3 \neq 1$
#3 *Remove 1 from* $\mathcal{D}(v_3)$	$\{3\}$	$\{1\}$	$\{2\}$	$(3,1,2)$ is a solution

How to solve a Sudoku with CP ?

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

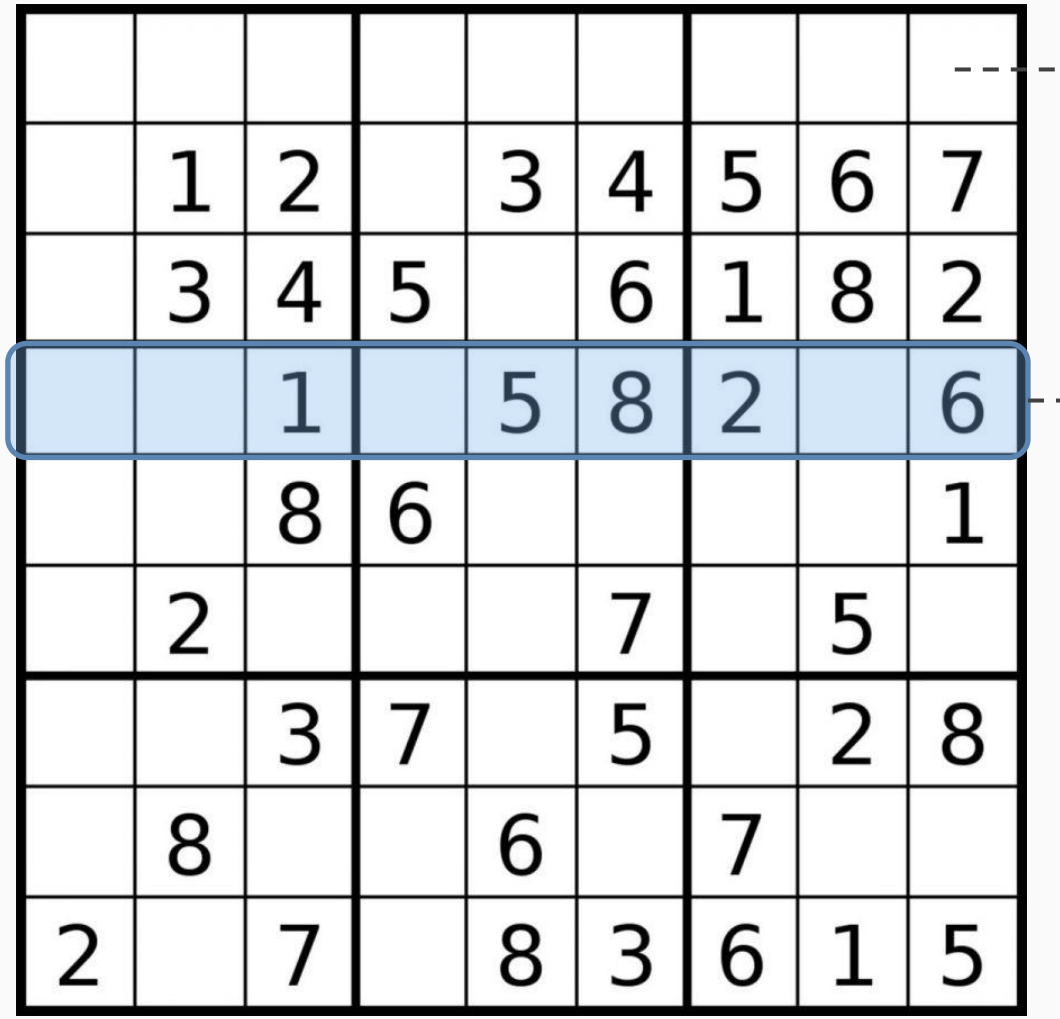
How to solve a Sudoku with CP ?

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

For each cell i, an integer variable:

- $x_i \in [1, 9]$ *if empty*
- $x_i = c$ *if not empty*

How to solve a Sudoku with CP ?



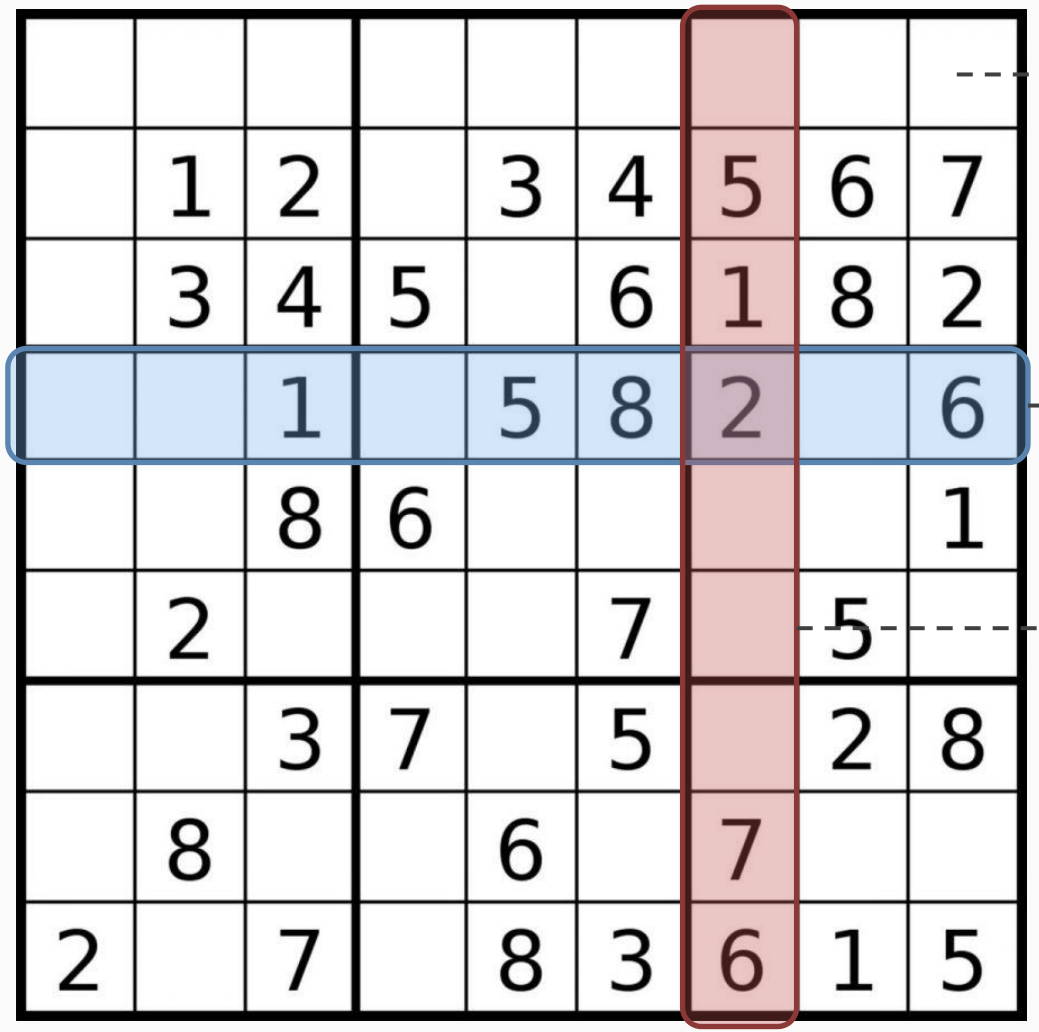
For each cell i, an integer variable:

- $x_i \in [1, 9]$ *if empty*
- $x_i = c$ *if not empty*

For each set of variables R_i *in a row:*

- *AllDifferent*(R_i)

How to solve a Sudoku with CP ?



For each cell i, an integer variable:

- $x_i \in [1, 9]$ *if empty*
- $x_i = c$ *if not empty*

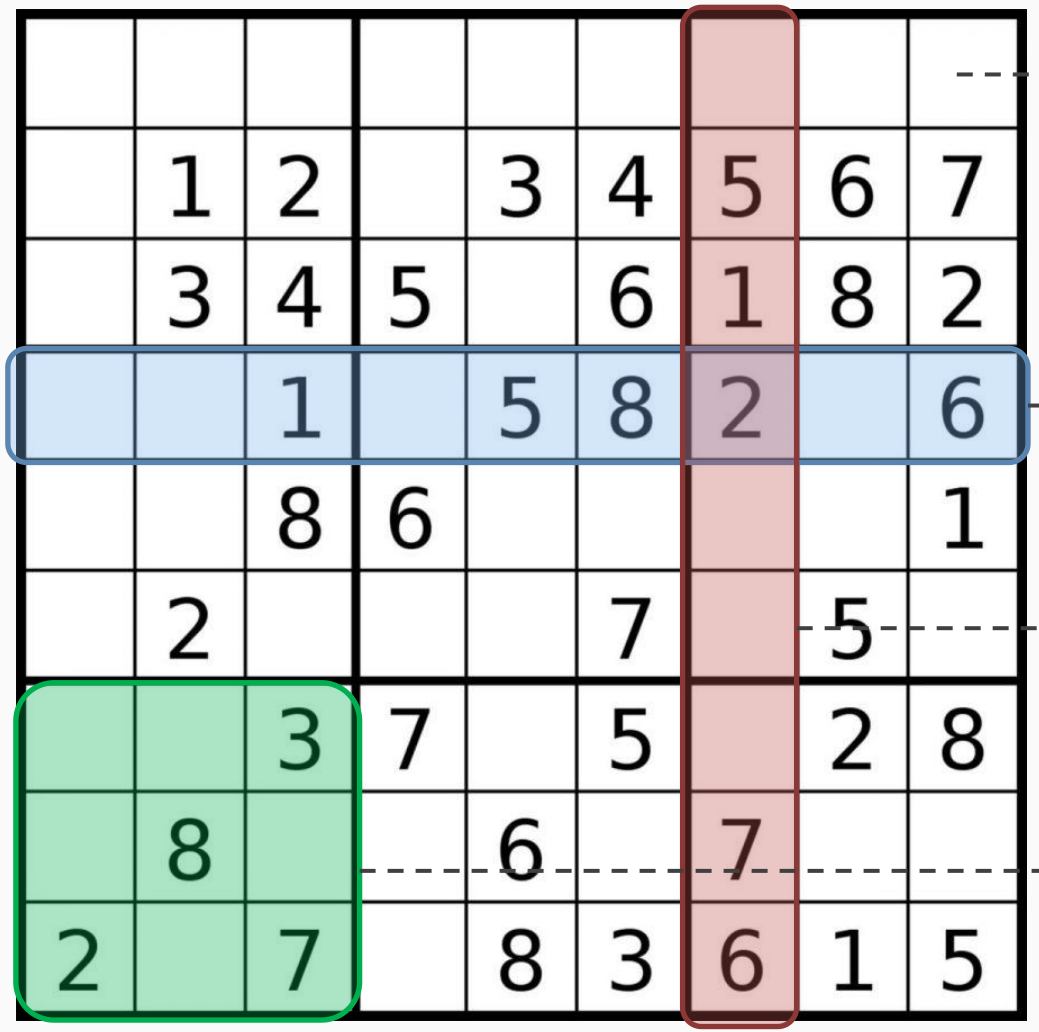
For each set of variables R_i *in a row:*

- *AllDifferent(*R_i*)*

For each set of variables C_i *in a column:*

- *AllDifferent(*C_i*)*

How to solve a Sudoku with CP ?



For each cell i, an integer variable:

- $x_i \in [1, 9]$ *if empty*
- $x_i = c$ *if not empty*

For each set of variables R_i *in a row:*

- *AllDifferent*(R_i)

For each set of variables C_i *in a column:*

- *AllDifferent*(C_i)

For each set of variables S_i *in a square:*

- *AllDifferent*(S_i)

Thank you !


AMAP
IRD
Institut de Recherche pour le Développement
FRANCE