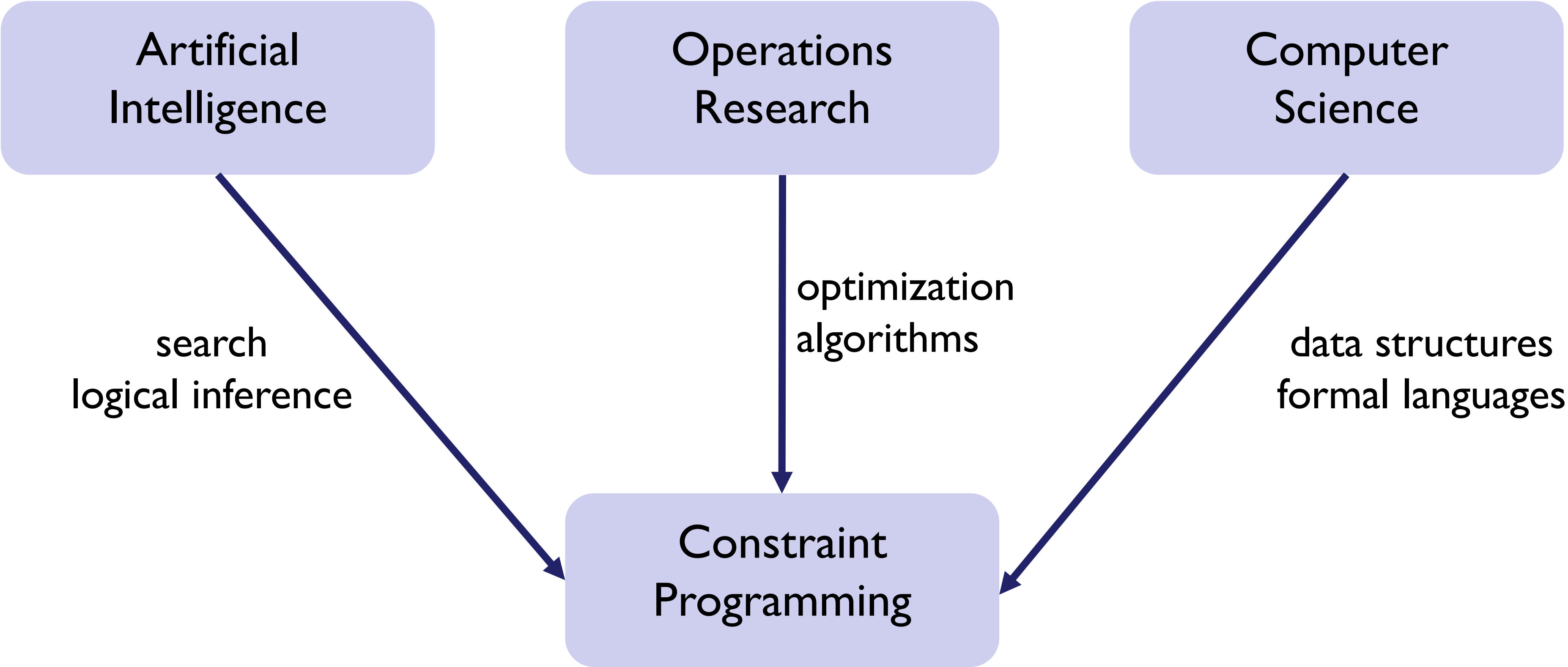
MTH4410
Constraint Programming

Merci à Willem-Jan van Hoeve, CMU.

Outline

- Successful Applications
- Modeling
- Solving
- Some details
  - global constraints
  - scheduling
- Integrated methods (MIP+CP)

Constraint Programming Overview



Artificial Intelligence

Operations Research

Computer Science

search
logical inference

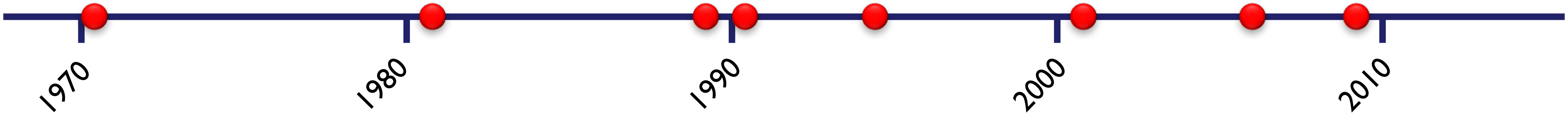
optimization
algorithms

data structures
formal languages

Constraint Programming

Evolution events of CP

- 1970s: Image processing applications in AI; Search+qualitative inference
- 1980s: Logic Programming (Prolog); Search + logical inference
- 1989: CHIP System; Constraint Logic Programming
- 1990s: Constraint Programming; Industrial Solvers (ILOG, Eclipse,…)
- 1994: Advanced inference for *alldifferent* and *resource scheduling*
- 2000s: Global constraints; integrated methods; modeling languages
- 2006: CISCO Systems acquires Eclipse CLP solver
- 2009: IBM acquires ILOG CP Solver & Cplex



Successful applications

Sport Scheduling

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

Schedule of 1997/1998 ACC basketball league (9 teams)

- various complicated side constraints
- all 179 solutions were found in 24h using enumeration and integer linear programming [Nemhauser & Trick, 1998]
- all 179 solutions were found in less than a minute using constraint programming [Henz, 1999, 2001]

Hong Kong Airport

- Gate allocation at the new (1998) Hong Kong airport
- System was implemented in only four months, includes constraint programming technology (ILOG)
- Schedules ~800 flights a day

  (47 million passengers in 2007)

G. Freuder and M. Wallace. Constraint Technology and the Commercial World. *IEEE Intelligent Systems* 15(1): 20-23, 2000.

Port of Singapore

Railroad Optimization

- Netherlands Railways has among the densest rail networks in the world, with 5,500 trains per day
- Constraint programming is one of the components in their railway planning software, which was used to design a new timetable from scratch (2009)
- Much more robust and effective schedule, and $75M additional annual profit
- INFORMS Edelman Award winner (2009)

Modeling in CP

CP Modeling basics

- CP models are very different from MIP models

- Virtually any expression over the variables is allowed
  - e.g., $x^3(y^2 - z) \geq 25 + x^2 \cdot \max(x,y,z)$

- CP models can be much more intuitive, close to natural language

- As a consequence, CP applies a different solving method compared to MIP

CP Variables

- Variables in CP can be the same as in your regular MIP model:
  - binary, integer, continuous

- In addition, they may take a value from *any* finite set
  - e.g., x in {a,b,c,d,e}
  - the set of possible values is called the *domain* of a variable

- Finally, there are some ‘special’ variable types for modeling ‘scheduling’ applications

CP Constraints

- A constraint is a relation between one or more variables.

- Let i and j be two integer variables

  i in {0..10};

  j in {0..10};

- Let R(i,j) be the following constraint

- When R(i,j) is asserted:
  - The domain for i is restricted to {1,2,5,7}
  - The domain for j is restricted to {2,3,4,10}

R

i	j
1	2
1	3
2	4
5	3
7	10

CP Constraints

- A solution to a constraint problem assigns a value to all the variables in such a way that all the constraints are satisfied

- i=2, j=4, k=8 is a solution of the system of three constraints R,S,T below

R

i	j
2	2
1	3
2	4
5	3
7	10

S

i	k
2	1
1	8
2	8
5	1
4	10

T

k	j
1	4
1	3
2	7
5	3
8	4

CP Constraints

What does a constraint do?

- Feasibility checking
  - can the constraint be satisfied given the domains of its variables
- Pruning
  - remove values from the domains if they do not appear in any solution of the constraint.

Constraint Propagation

- When the domain of a variable is reduced, constraints may imply domain reductions for other related variables.

- Example:
  - Remove 1 from the domain of i

R

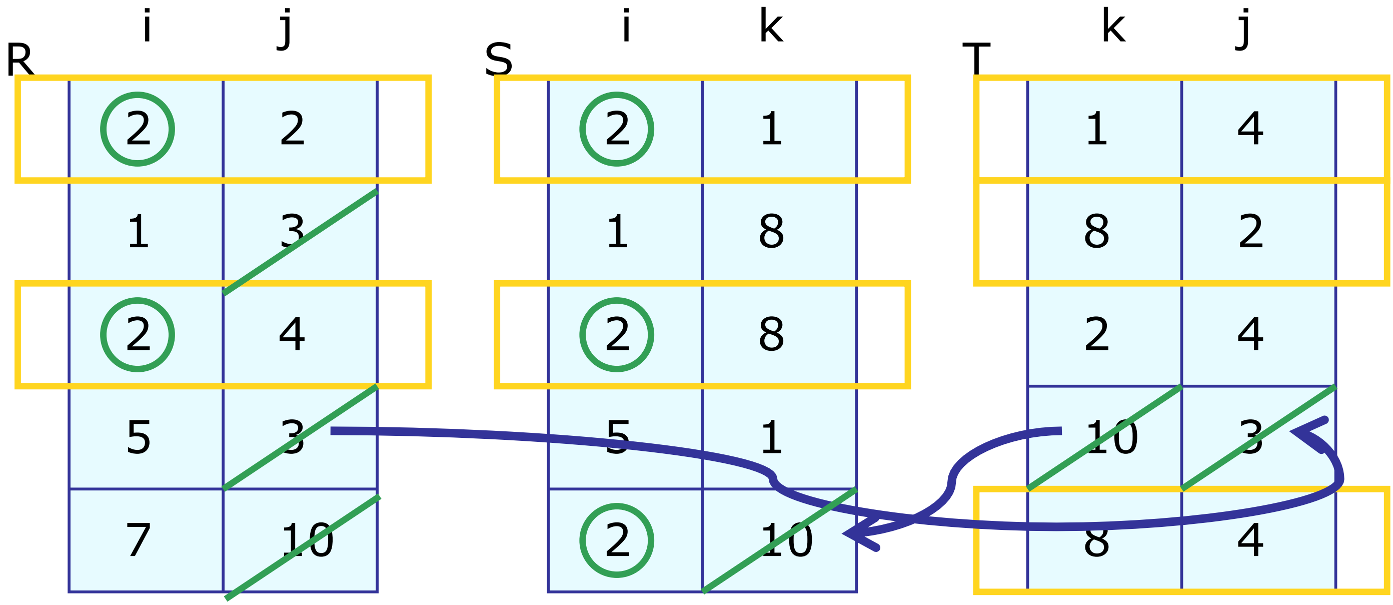
i	j
~~1~~	~~2~~
~~1~~	~~3~~
2	4
5	3
7	10

  - It results in removing 2 from the domain of j
  - The value 3 is still in the domain of j

Constraint Propagation

- When the domain of a variable is reduced, the effects of this change are propagated through all the constraints

- In this example, let us set i to the value 2



R

i	j
2	2
1	~~3~~
2	4
5	~~3~~
7	~~10~~

S

i	k
2	1
1	8
2	8
5	1
2	~~10~~

T

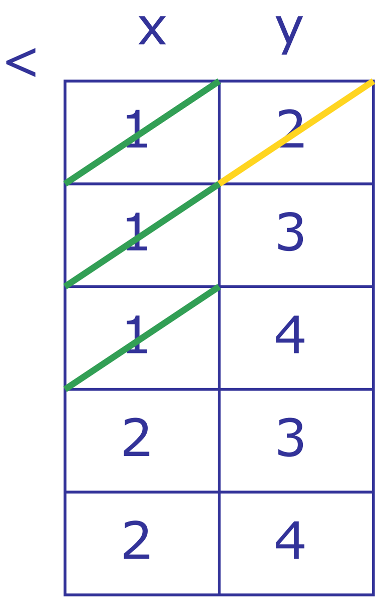
k	j
1	4
8	2
2	4
~~10~~	~~3~~
8	4

Constraints as Algorithms

- In most cases, it is inefficient to implement constraints using actual relational tables.

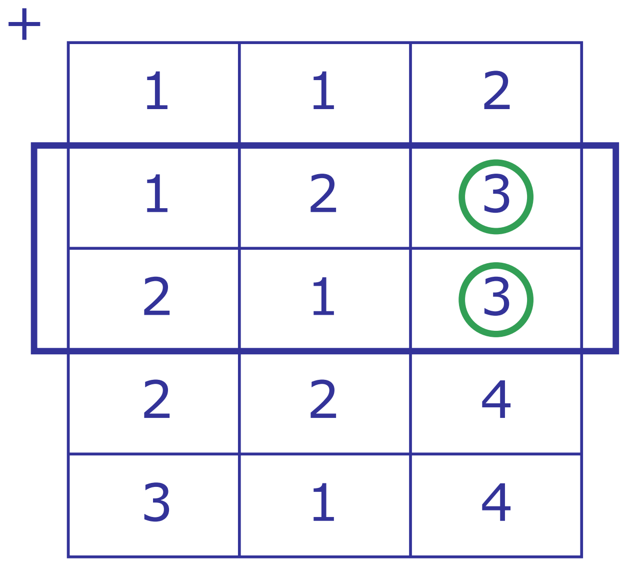
- CP languages thus use propagation algorithms to implement arithmetic constraints and all others.

- The propagation algorithm must behave in the same way as the corresponding extensional relation.



<

x	y
~~1~~	~~2~~
~~1~~	3
~~1~~	4
2	3
2	4



+

1	1	2
1	2	3
2	1	3
2	2	4
3	1	4

Example: Magic Series

- A series $S = (S_0,\ldots,S_n)$ is magic if S_i is the number of occurrences of i in S

0	1	2	3	4
?	?	?	?	?

Example: Magic Series

- A series S = (S_0,…,S_n) is magic if S_i is the number of occurrences of i in S

0	1	2	3	4
2	1	2	0	0

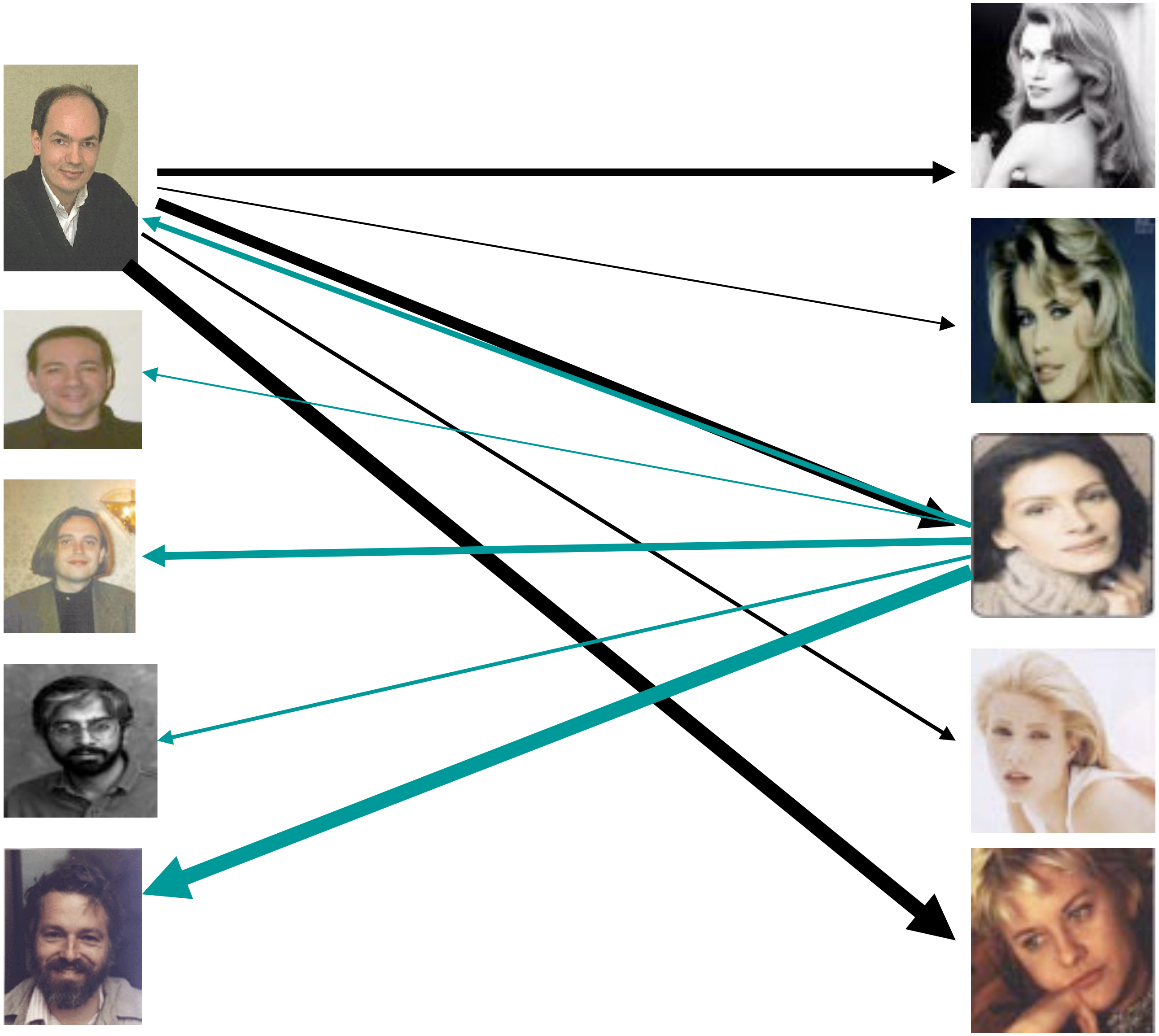
Reification

Reification

```
n = 5
D = {0..n-1}
var s[D] in D
forall(k in D) s[k] == sum(i in D) (s[i]==k));
```

- Reification
  - Allow constraints inside constraints
  - Replace the constraint in () by a 0/1 variables representing the truth value of the constraint

Example: Stable Marriages



Example: Stable Marriages

- A marriage is stable between James and Kathryn provided that
  - Whenever James prefers another woman, say Anne, to Kathryn, then Anne prefers her husband to James;
  - Whenever Kathryn prefers another man, say Laurent, to James, then Laurent prefers his spouse to Kathryn.

Example: Stable Marriages

```
Men = {Richard,James,John,Hugh,Greg}
Women = {Helen,Tracy,Linda,Sally,Wanda}
preferm[Men,Women] = …
preferw[Women,Men] = …
var wife[Men] in {Women}
var husband[Women] in {Men}

forall(i in Men) husband[wife[i]] == i

forall(i in Women) wife[husband[i]] == i

forall(i in Men,j in Women)
 (preferm[i,j] > preferm[i,wife[i]]) => (preferw[j,husband[j]] > preferw[j,i])

forall(i in Men,j in Women)
 (preferw[j,i] < preferw[j,husband[j]]) => (preferm[i,wife[i]] < preferm[i,j]);
}
```

Element Constraint

Reification

Implication

Element Constraints

- Element constraints
  - ability to index an array/matrix with a decision variable or an expression;
- Logical constraints
  - ability to express any logical combination of constraint
  - see also reification

The Element Constraint

- X : variable

~~3~~	~~4~~	5

- Y : variable

~~0~~	~~1~~	2	3	~~4~~	~~5~~
[0]	[1]	[2]	[3]	[4]	[5]

- C : array

3	4	5	5	4	3

- Constraint: X = C[Y]
- X ≠ 3
- Y ≠ 1 & Y ≠ 4

The Element Constraint

- Facility location: want a constraint that customer c can be assigned to warehouse i only if warehouse open. (open[i]=1 if warehouse i is open)

- MIP: x[c,i] is 1 if customer c is assigned to i

  x[c,i] <= open[i]

- CP: w[c] is the warehouse customer c is assigned to

  open[w[c]] = 1; (not a 0,1 variable)

Assignment Problem

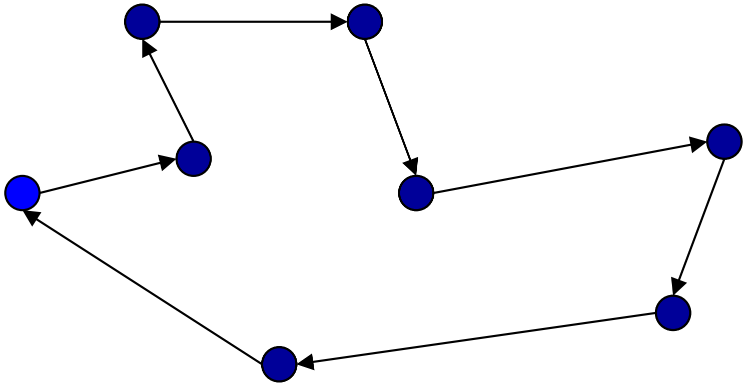
- Solve the following assignment problem with AIMMS
  - Given 5 tasks (t_1 to t_5) and 5 employees (e_1 to e_5)
  - Assign one and only one task to each employees such that the assignment minimizes the following costs:

T\E	1	2	3	4	6
1	2	3	5	1	8
2	3	4	3	4	5
3	1	3	4	7	9
4	3	3	2	6	4
5	5	7	2	8	5

  - Can you compare with a MIP version of this problem ?

Another example of Element: the TSP

- The traveling salesperson problem asks to find a closed tour on a given set of *n* locations, with minimum total length (see class on heuristics)
- Input: set of locations and distance d_{ij} between two locations i and j



TSP: MIP model

- Classical model based on ‘assignment problem’
- Binary variable x_{ij} represents whether the tour goes from i to j
- Objective

$$\min \ \sum_{ij} d_{ij} x_{ij}$$

- Need to make sure that we leave and enter each location exactly once

$$\sum_j x_{ij} = 1 \text{ for all } i$$

$$\sum_i x_{ij} = 1 \text{ for all } j$$

- Remove all possible subtours: there are exponentially many; impossible to model concisely in MIP
- MIP Solvers therefore resort to specialized solving methods for the TSP

TSP: CP model

- Variable x_i represents the i-th location that the tour visits (variable domain is {1,2,…,n})
- Objective

min $$d_{x_n, x_1} + \sum_{i=1}^{n-1} d_{x_i, x_{i+1}}$$

Another way to write Element constaints is to put variables as subscripts!

- Constraint

alldifferent(x_1, x_2, …, x_n)

this is a ‘global’ constraint

Example: Alldifferent

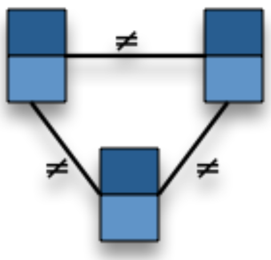
Alldifferent$(x_1,x_2,\ldots,x_n)$ semantically equivalent to

$\{ x_i \neq x_j$ for all $i \neq j \}$

Model 1: $x_1 \in \{a,b\}, x_2 \in \{a,b\}, x_3 \in \{a,b,c\}$

$x_1 \neq x_2, x_1 \neq x_3 , x_2 \neq x_3$

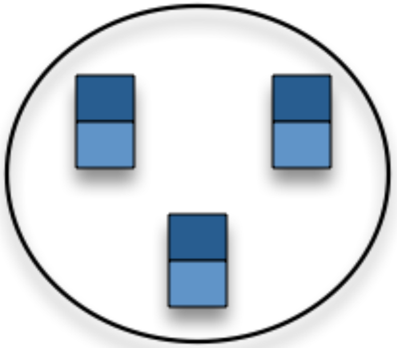
→ no domain values will be filtered



Model 2: $x_1 \in \{a,b\}, x_2 \in \{a,b\}, x_3 \in \{a,b,c\}$

alldifferent(x_1,x_2,x_2)

→ global view of *alldifferent*: $x_3 \in \{c\}$



Grouping constraints together allows more domain filtering!

Filtering for alldifferent

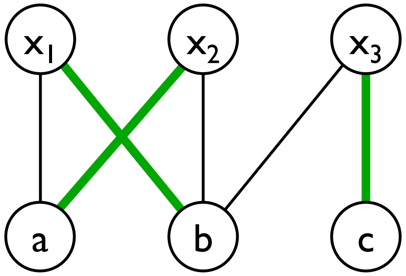
Observation [Régin, 1994]:

solution to *alldifferent* ⇔ matching in bipartite graph covering all variables

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$, $x_3 \in \{\cancel{b},c\}$

alldifferent(x_1,x_2,x_3)



Filtering: remove all edges (and corresponding domain values) that are not in any matching covering the variables

Find initial matching: $O(m\sqrt{n})$ time[1] [Hopcroft and Karp, 1973]

Filter all inconsistent edges?

[1] for n variables and m edges

MIP and CP model compared

- The CP model needs only n variables, while the MIP model needs n^2 variables (n is #locations)

- The MIP model is of exponential size, while the CP model only needs one single constraint

- The CP model is more intuitive, as it is based directly on the problem structure: the ordering of the locations in the tour

Note: The special-purpose MIP solving methods outperform CP on *pure* TSP. In presence of side constraints (e.g., time windows), CP becomes competitive.

Illustration: Sudoku

- each row contains numbers 1 up to 9
- each column contains numbers 1 up to 9
- each block contains numbers 1 up to 9

Sudoku *puzzle*:

try to complete partially filled square

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

CP model for Sudoku

variables and domains:

x_{ij} in {1,2,3,4,5,6,8,9} for all i,j in 1..9

constraints:

alldifferent(x_{ij} : j=1..9) for all rows i

alldifferent(x_{ij} : i=1..9) for all columns j

alldifferent(x_{ij} : i,j in block b) for all blocks b

x_{ij} = k if cell (i,j) is pre-set to value k

See Sudoku.aimmspack

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

Solving time

Experimental results over larger Sudoku instances (16×16) [1]

not-equal constraints

$\{ x_i \neq x_j \text{ for all } i \neq j \}$

solved: 94%

total time: 249.21s

backtracks: 2,284,716

alldifferent constraints

alldifferent(x_{ij})

solved: 100%

total time: 6.47s

backtracks: 3020

What is the effect of changing the inference level from ‘default’ to ‘extended’ in our AIMMS model?

[1] time limit 600s

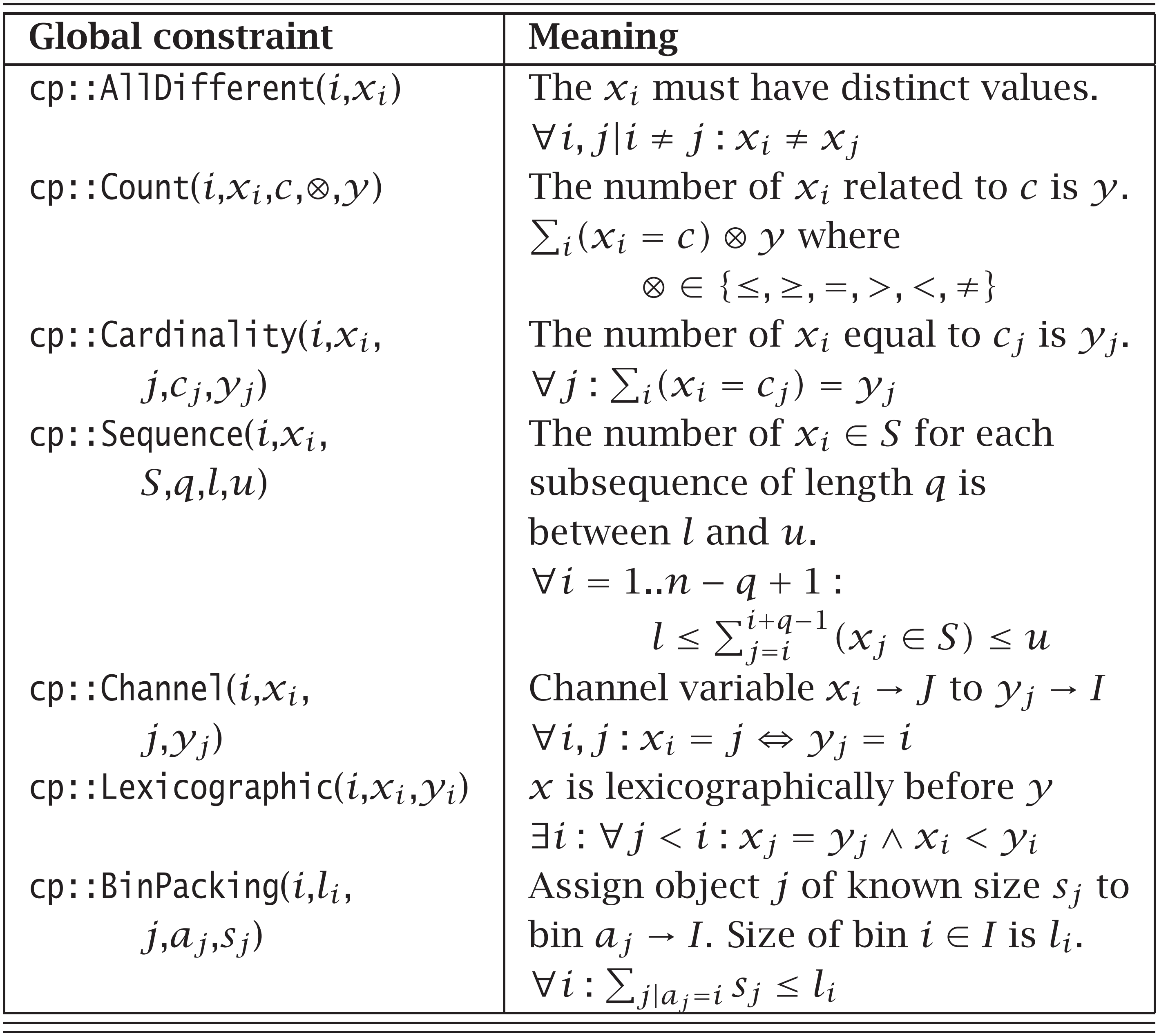
Global Constraints

- Examples
  - *Alldifferent, Count, BinPacking, SequentialSchedule, ParallelSchedule, NetworkFlow, …*

- Global constraints represent combinatorial structure
  - Can be viewed as the combination of elementary constraints
  - Expressive building blocks for modeling applications
  - Embed powerful algorithms from OR, Graph Theory, AI, CS, …

- Essential for the successful application of CP
  - When modeling a problem, always try to identify possible global constraints that can be used

List of Global Constraints (in AIMMS)



Global constraint	Meaning
`cp::AllDifferent(`i,x_i`)`	The x_i must have distinct values. $\forall i,j \vert i \neq j : x_i \neq x_j$
`cp::Count(`i,x_i,c,$\otimes$,y`)`	The number of x_i related to c is y. $\sum_i (x_i = c) \otimes y$ where $\otimes \in \{\leq, \geq, =, >, <, \neq\}$
`cp::Cardinality(`i,x_i, j,c_j,y_j`)`	The number of x_i equal to c_j is y_j. $\forall j : \sum_i (x_i = c_j) = y_j$
`cp::Sequence(`i,x_i, S,q,l,u`)`	The number of $x_i \in S$ for each subsequence of length q is between l and u. $\forall i = 1..n - q + 1 :$ $l \leq \sum_{j=i}^{i+q-1} (x_j \in S) \leq u$
`cp::Channel(`i,x_i, j,y_j`)`	Channel variable $x_i \to J$ to $y_j \to I$ $\forall i,j : x_i = j \Leftrightarrow y_j = i$
`cp::Lexicographic(`i,x_i,y_i`)`	x is lexicographically before y $\exists i : \forall j < i : x_j = y_j \wedge x_i < y_i$
`cp::BinPacking(`i,l_i, j,a_j,s_j`)`	Assign object j of known size s_j to bin $a_j \to I$. Size of bin $i \in I$ is l_i. $\forall i : \sum_{j \vert a_j = i} s_j \leq l_i$

Summary of CP modeling

- Variables range over finite or continuous domain:

  $v \in \{a,b,c,d\}$, $start \in \{0,1,2,8,9,10\}$, $z \in [2.18, 4.33]$

- Algebraic expressions:

  $x^3(y^2 - z) \geq 25 + x^2 \cdot \max(x,y,z)$

- Variables as subscripts:

  $y = cost[x]$ (here y and x are variables, 'cost' is an array of parameters)

- Reasoning with meta-constraints:

  $\sum_i (x_i > T_i) \leq 5$

- Logical relations in which (meta-)constraints can be mixed:

  $((x < y) \text{ OR } (y < z)) \Rightarrow (c = \min(x,y))$

- Global constraints (a.k.a. symbolic constraints):

  Alldifferent$(x_1, x_2, \ldots, x_n)$

  SequentialSchedule$([start_1,\ldots, start_n], [dur_1,\ldots,dur_n], [end_1,\ldots,end_n])$

CP Solving

CP Solving

In general

- CP variables are
  - discrete (i.e., integer valued)
- while CP constraints are
  - non-linear
  - non-differentiable
  - discontinuous

Hence, no traditional Operations Research technique can solve these models (LP, NLP, MIP, etc)

Basics of CP solving

- CP solving is based on intelligently enumerating all possible variable-value combinations
  - called backtracking search
  - similar to branch&bound for MIP

- Unlike branch&bound, CP does not solve a LP relaxation at each search node, but applies specific *constraint propagation* algorithms

- These propagation algorithms are applied to individual constraints, and their role is to limit the size of the search tree

Solving

Example:

variables/domains	$x_1 \in \{1,2\}$, $x_2 \in \{0,1,2,3\}$, $x_3 \in \{2,3\}$
constraints	$x_1 > x_2$
	$x_1 + x_2 = x_3$
	alldifferent(x_1,x_2,x_3)

Solving

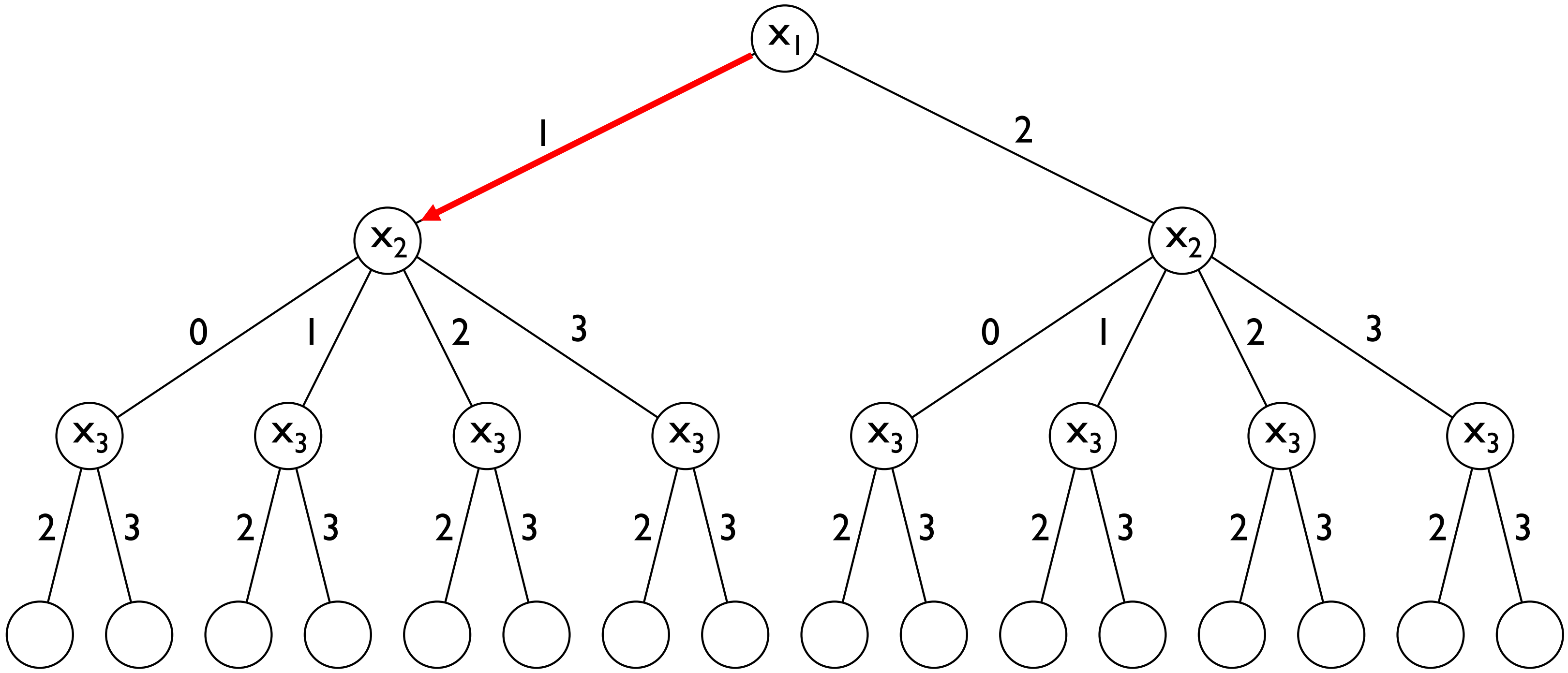
Example:

variables/domains $x_1 \in \{1,2\}$, $x_2 \in \{0,1,\cancel{2},\cancel{3}\}$, $x_3 \in \{2,3\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)



Solving

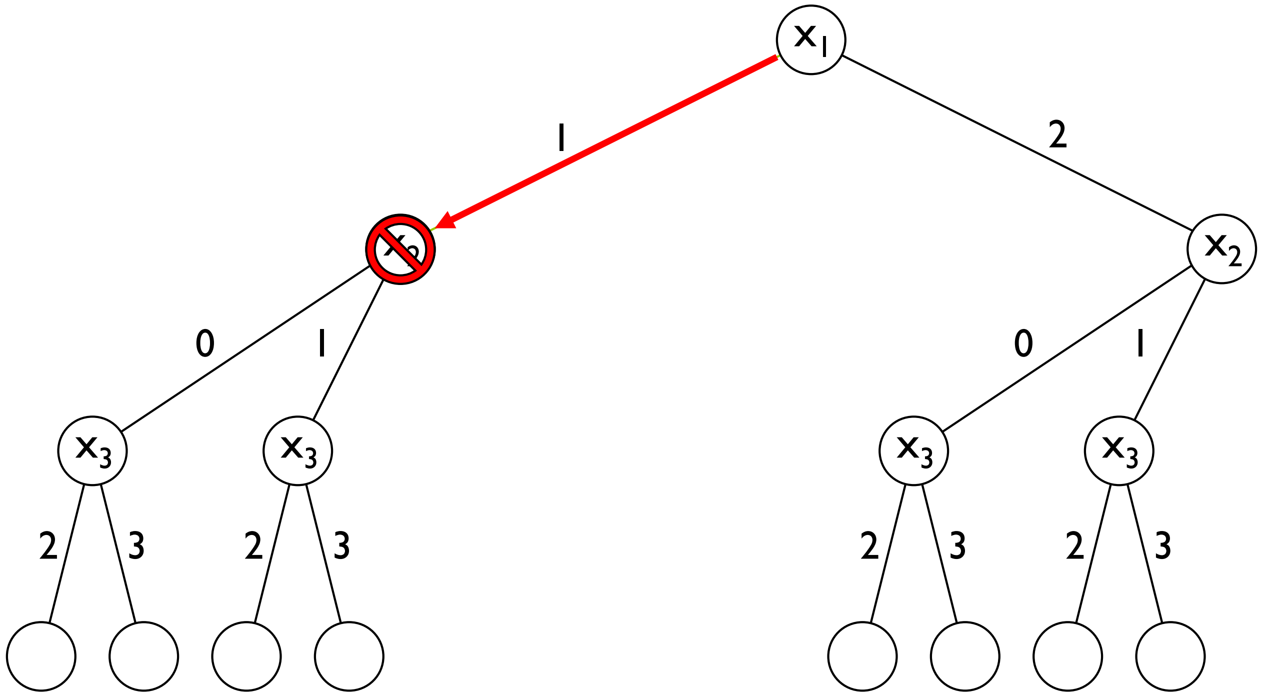
Example:

variables/domains $x_1 \in \{\cancel{1}\}$, $x_2 \in \{\cancel{0},\cancel{1}\}$, $x_3 \in \{\cancel{2},\cancel{3}\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)



Solving

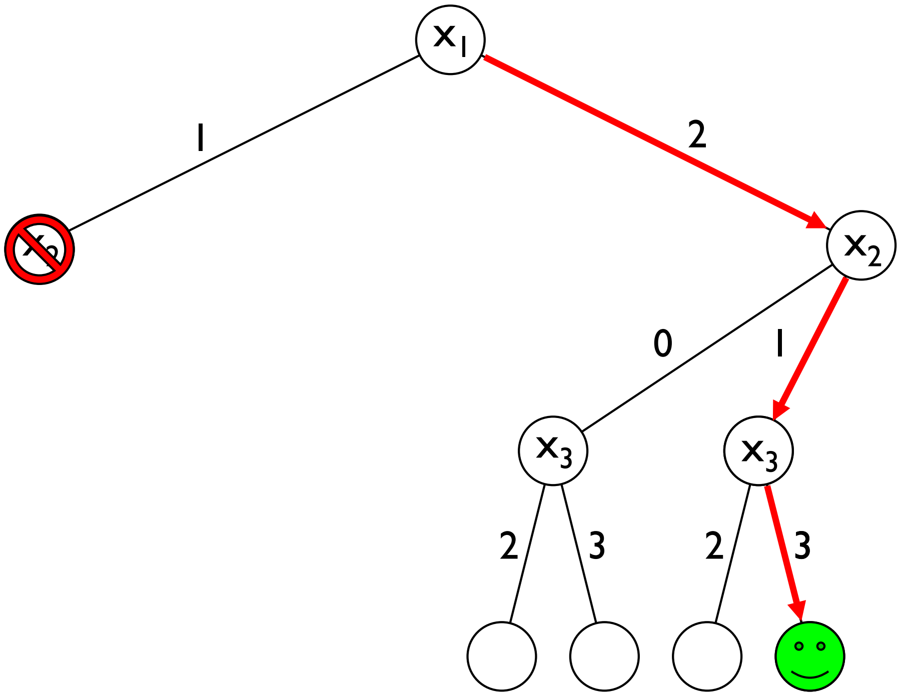
Example:

variables/domains $x_1 \in \{2\}, x_2 \in \{\cancel{0},1\}, x_3 \in \{\cancel{2},3\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

$\textit{alldifferent}(x_1,x_2,x_3)$



CP - Summary

The solution process of CP interleaves

- Domain filtering
  - remove inconsistent values from the domains of the variables, based on individual constraints
- Constraint propagation
  - propagate the filtered domains through the constraints, by re-evaluating them until there are no more changes in the variable domains
- Search
  - implicitly all possible variable-value combinations are enumerated, but the search tree is kept small due to the domain filtering and constraint propagation

Another example

Partial Latin Square (order 3)

- A number in {1,2,3} in each cell
- Numbers on each row must be pairwise different
- Numbers on each column must be pairwise different
- Some cells are pre-filled

3	1	2
1	2	3
2	3	1

A possible solution

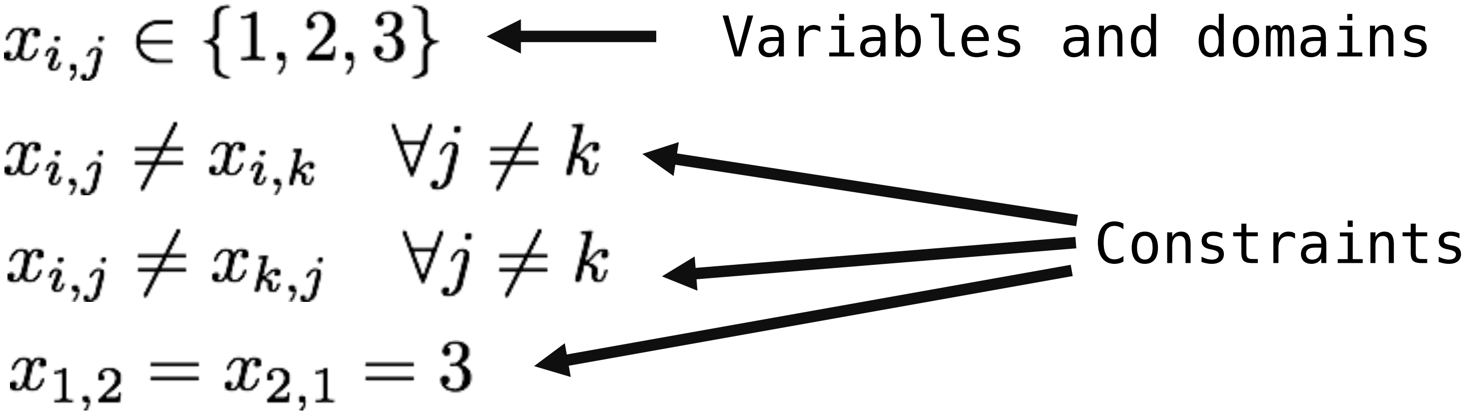
Another example

Partial Latin Square (order 3)

- A number in {1,2,3} in each cell
- Numbers on each row must be pairwise different
- Numbers on each column must be pairwise different
- Some cells are pre-filled

3	1	2
1	2	3
2	3	1

As a CSP:



$x_{i,j} \in \{1, 2, 3\}$

Variables and domains

$x_{i,j} \neq x_{i,k} \quad \forall j \neq k$

$x_{i,j} \neq x_{k,j} \quad \forall j \neq k$

$x_{1,2} = x_{2,1} = 3$

Constraints

Another example

Partial Latin Square (order 3)

- A number in {1,2,3} in each cell
- Numbers on each row must be pairwise different
- Numbers on each column must be pairwise different
- Some cells are pre-filled

3	1	2
1	2	3
2	3	1

As a CSP:

$$x_{i,j} \in \{1,2,3\}$$

$x_{i,j} \neq x_{i,k} \quad \forall j \neq k$ ← Pairwise different on rows

$x_{i,j} \neq x_{k,j} \quad \forall j \neq k$ ← Pairwise different on cols

$x_{1,2} = x_{2,1} = 3$ ← Pre-filled cells

Another example

Before propagation

{1,2,3}	{1,2,3}	{1,2,3}
{1,2,3}	{1,2,3}	**3**
{1,2,3}	**3**	{1,2,3}

Another example

After propagation

{1,2,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	**3**
{1,2,~~3~~}	**3**	{1,2,~~3~~}

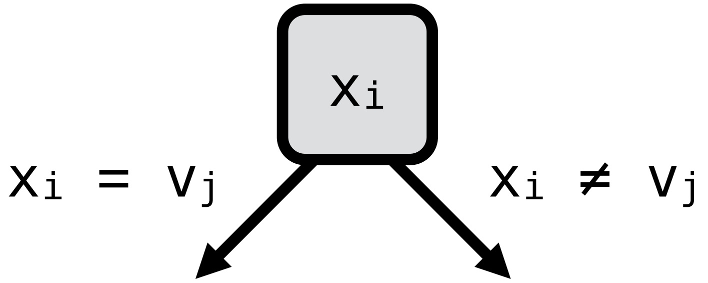
Another example

How to search for a solution?

The simplest approach is using Depth First Search

- Open a choice point
- On each branch post a new constraint
- So as to partition the solution space

A typical example:



- Choose a variable x_i
- Choose a value v_j in D_i
- Post $x_i = v_j$ on the left branch
- Post the opposite constraint on backtrack

] binary choice point

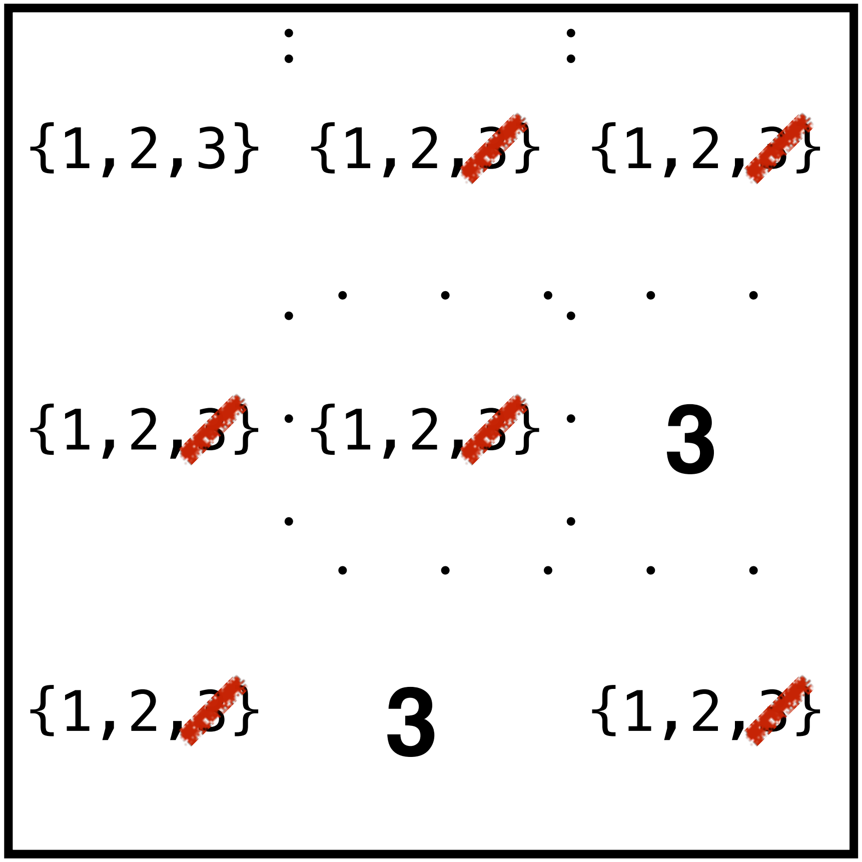
Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored

Let's see that in action:

- Choose var with smallest index
- Choose smallest value



{1,2,3} {1,2,~~3~~} {1,2,~~3~~}

{1,2,~~3~~} {1,2,~~3~~} 3

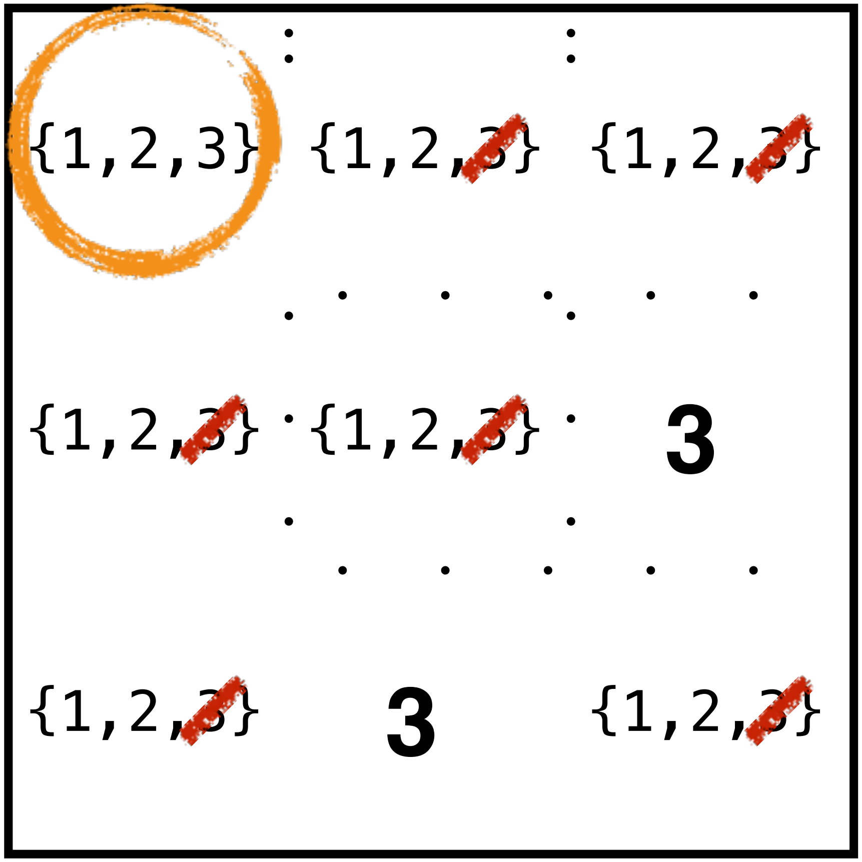
{1,2,~~3~~} 3 {1,2,~~3~~}

Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored

X_{11}

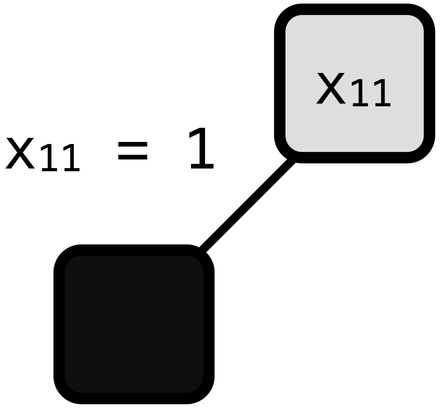


{1,2,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

Another example

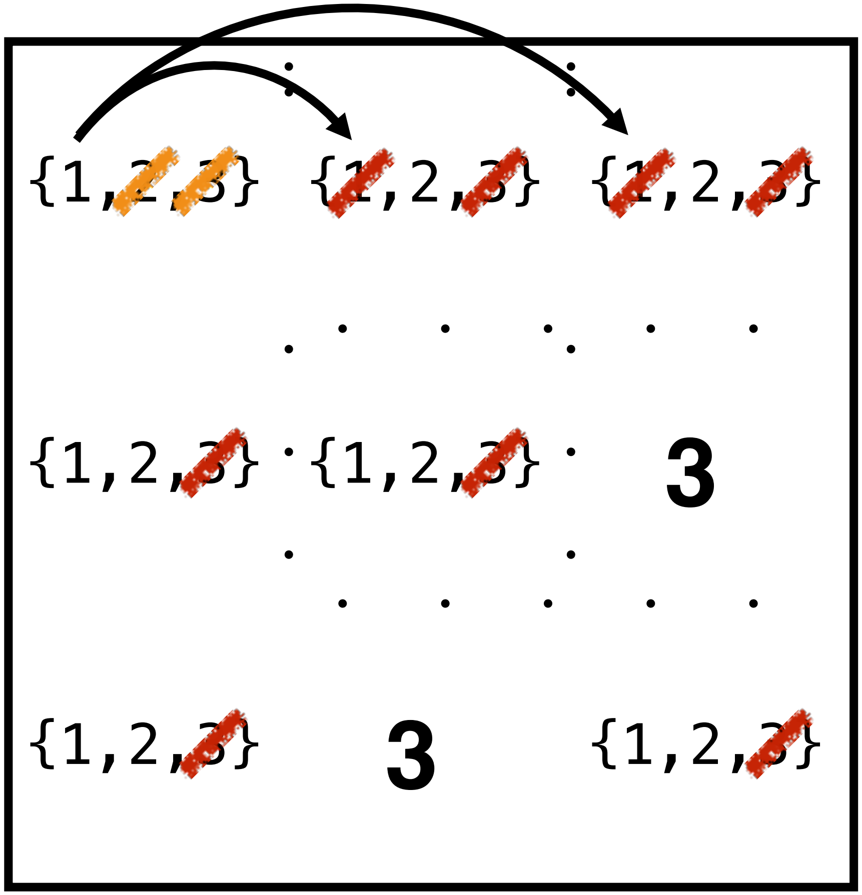
Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



$x_{11} = 1$

x_{11}

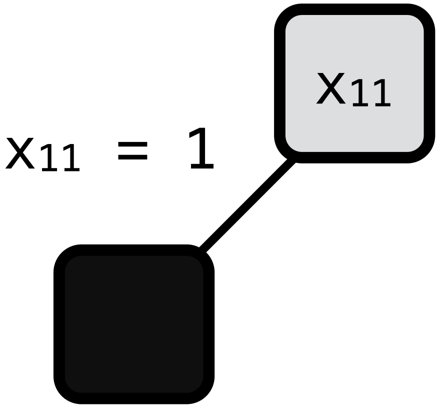


{1,~~2~~,~~3~~}	{~~1~~,2,~~3~~}	{~~1~~,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

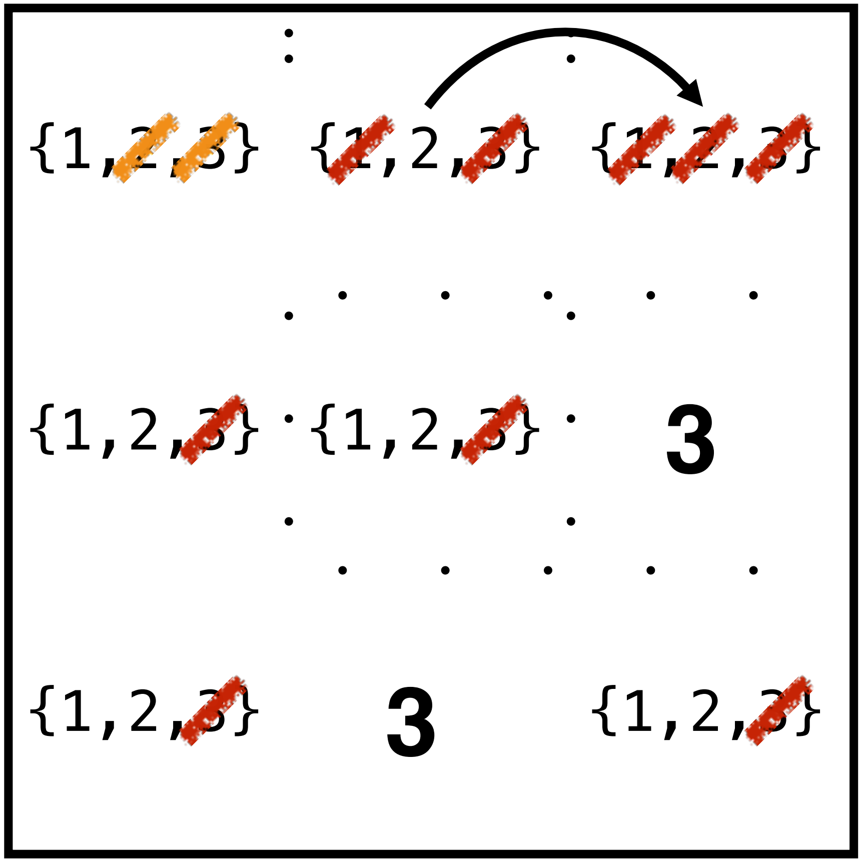
Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



domain wipeout

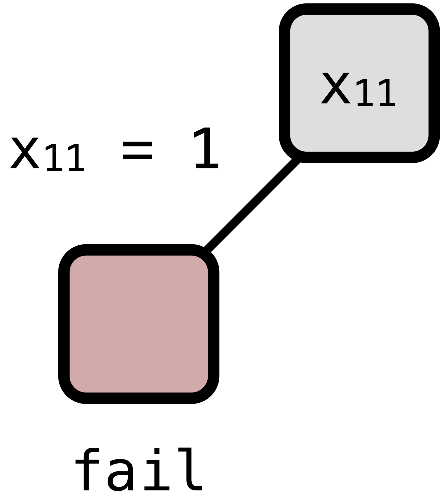


{1,~~2~~,~~3~~}	{~~1~~,2,~~3~~}	{~~1~~,~~2~~,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	**3**
{1,2,~~3~~}	**3**	{1,2,~~3~~}

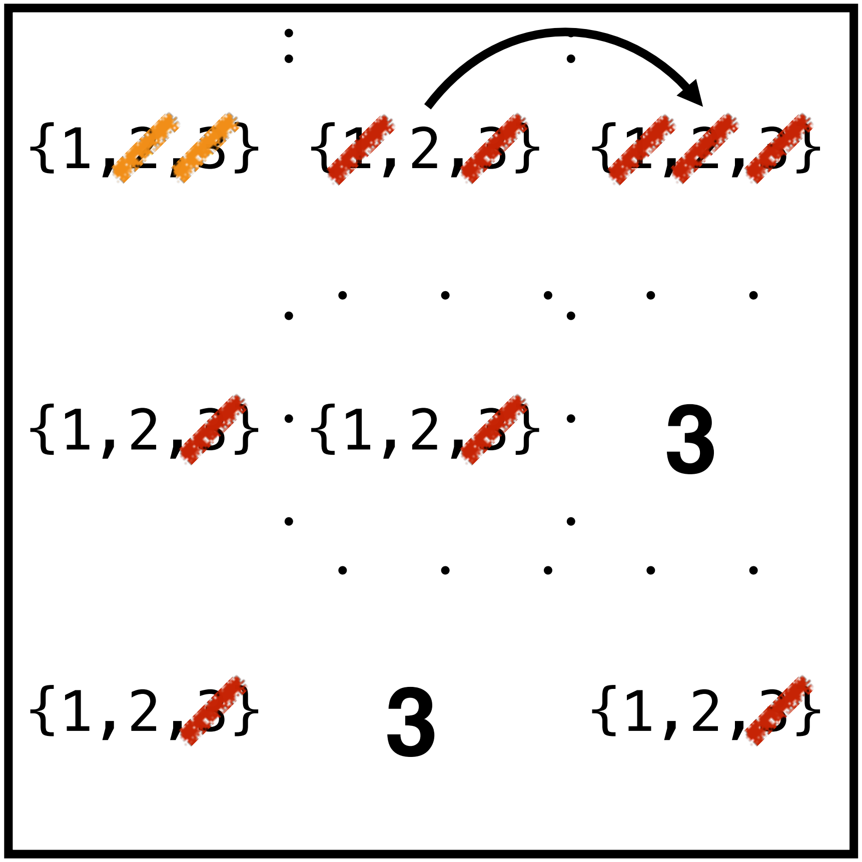
Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



domain wipeout

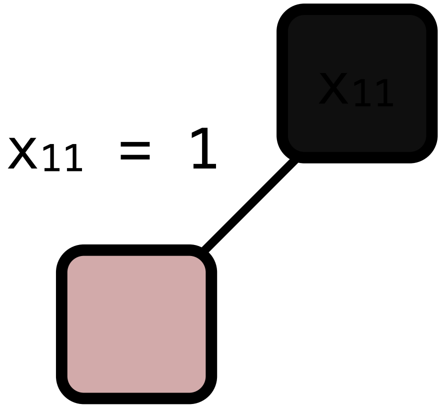


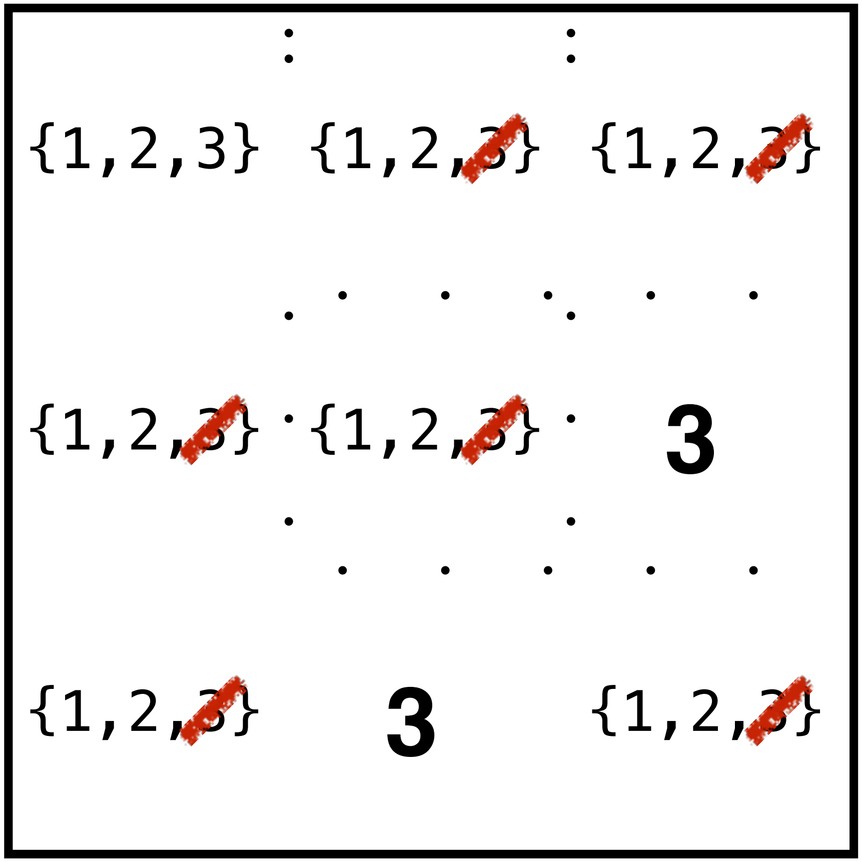
{1,~~2~~,~~3~~}	{~~1~~,2,~~3~~}	{~~1~~,~~2~~,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



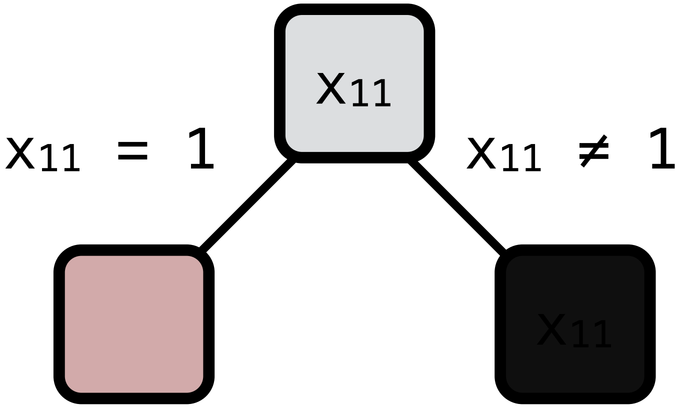


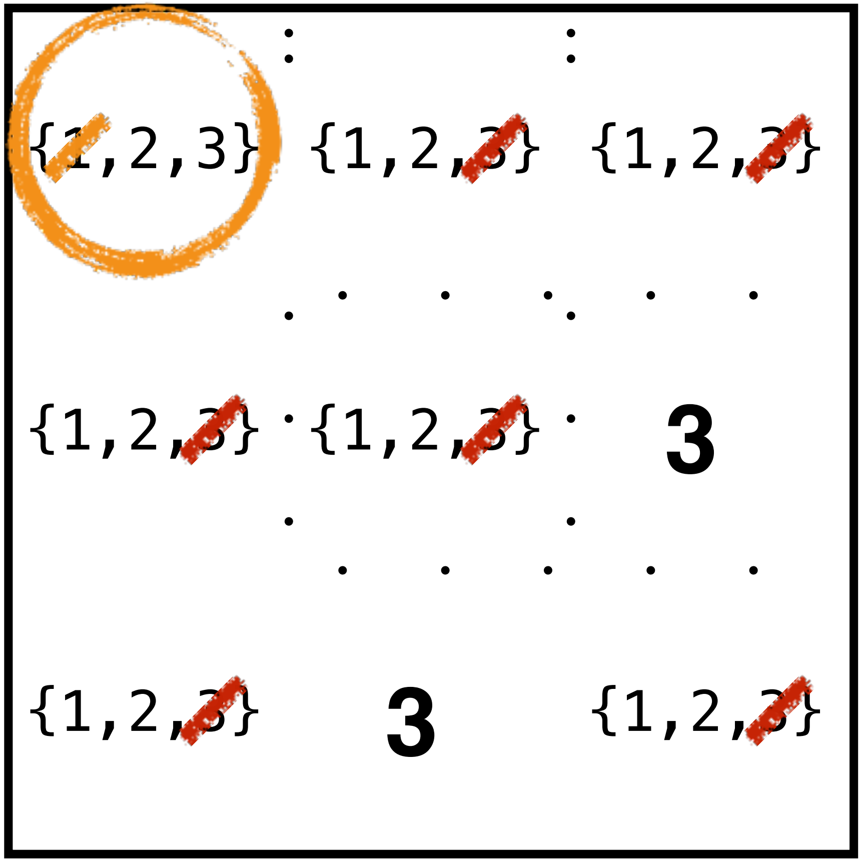
{1,2,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



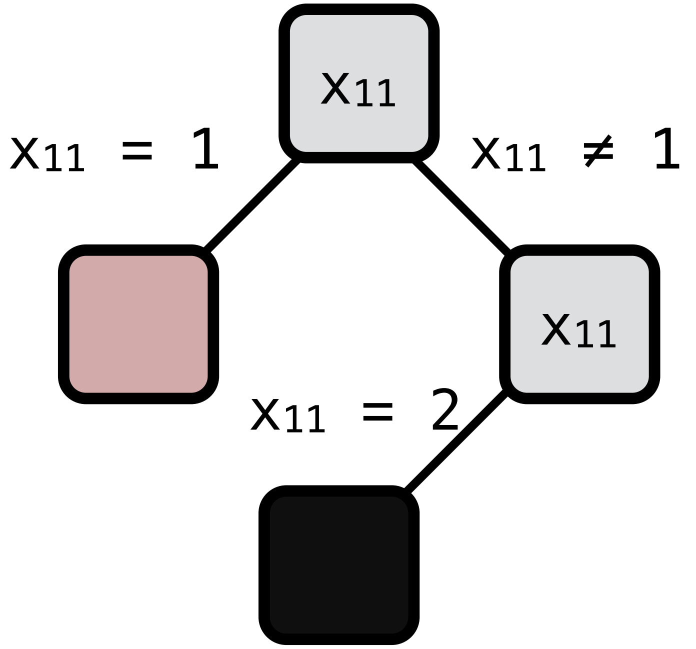


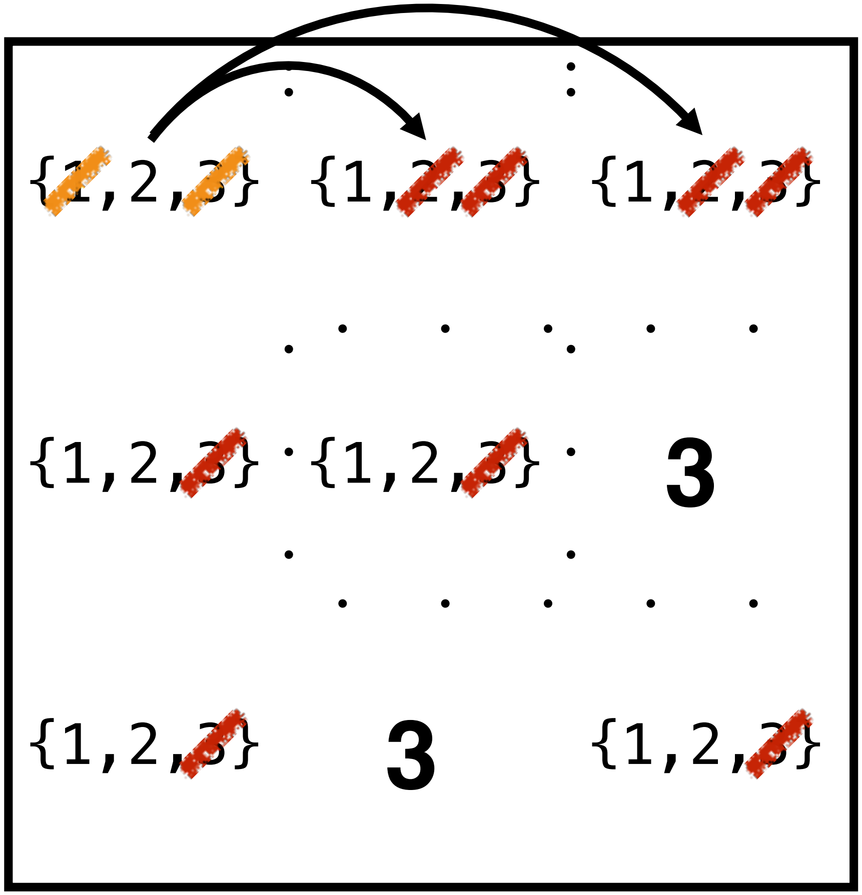
{~~1~~,2,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



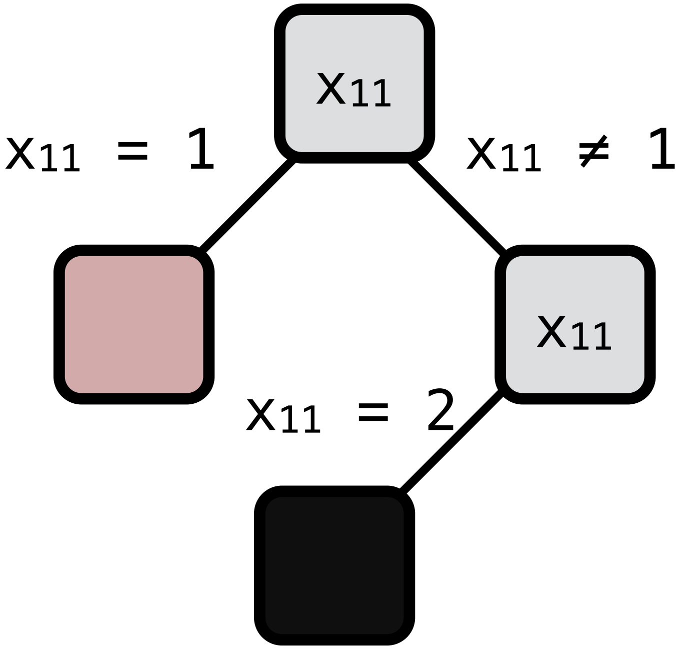


{~~1~~,2,~~3~~}	{1,~~2~~,~~3~~}	{1,~~2~~,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

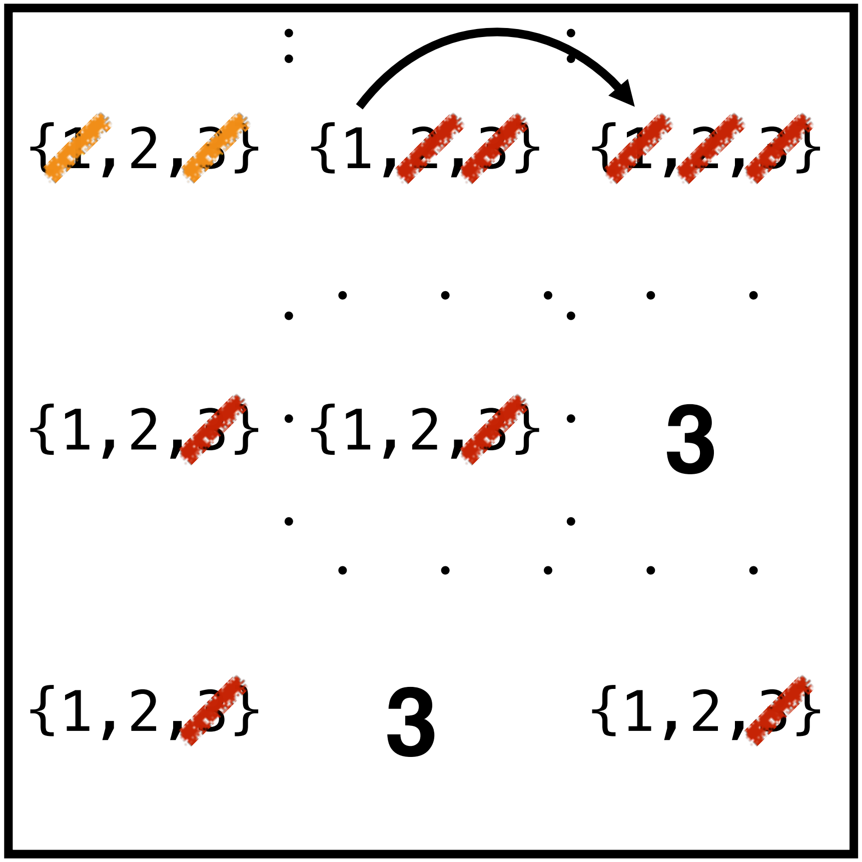
Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



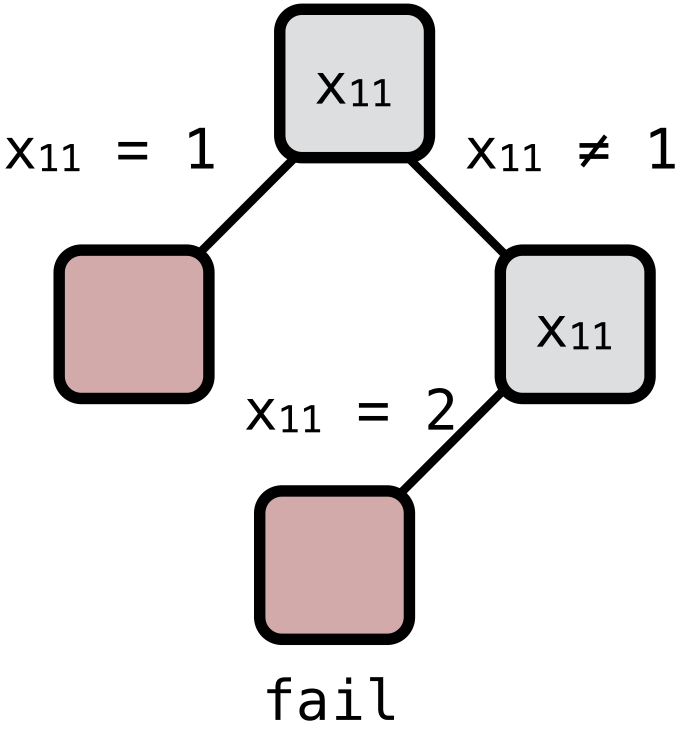
another wipeout



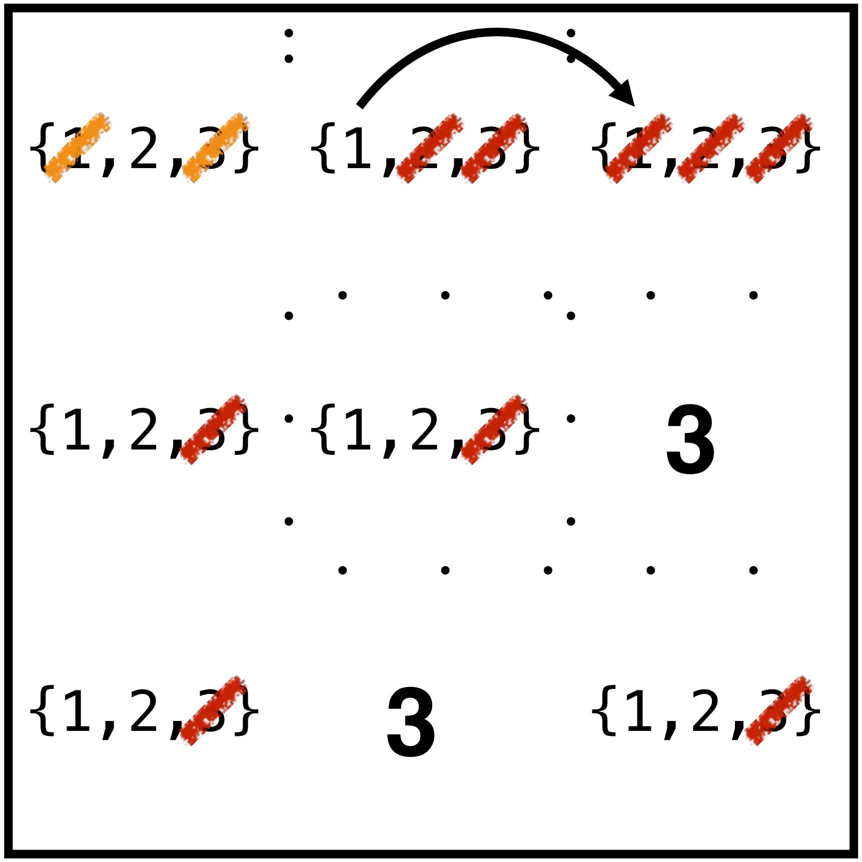
Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



another
wipeout

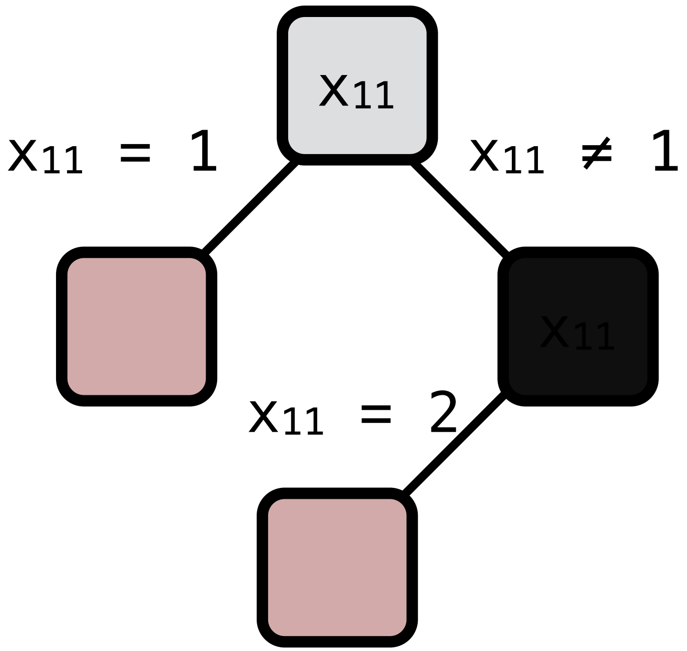


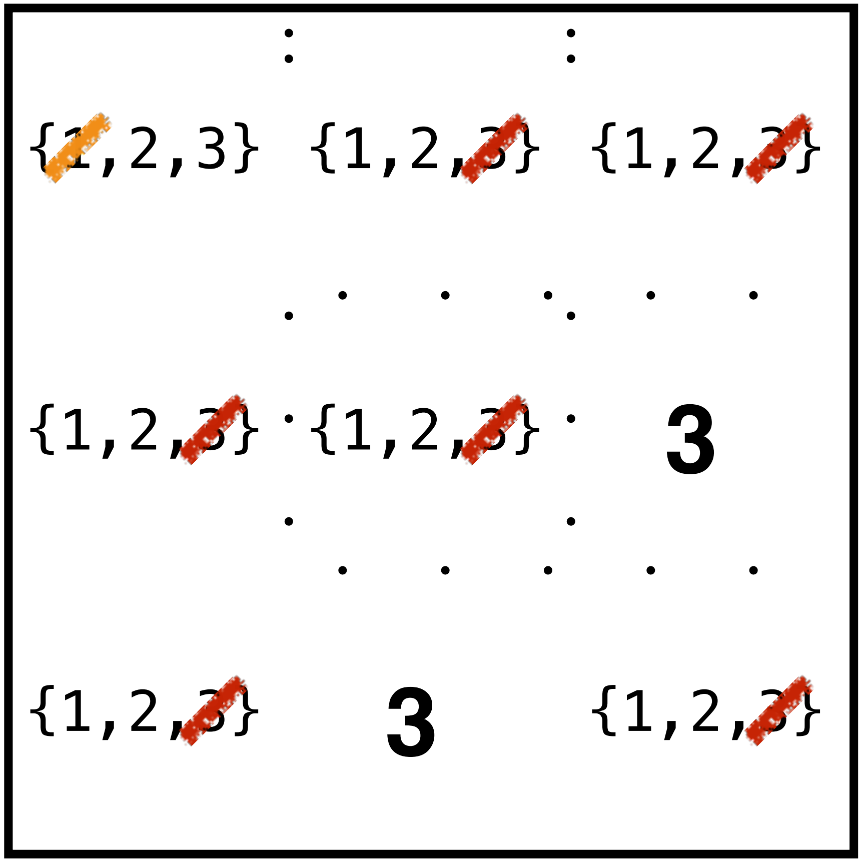
{~~1~~,2,~~3~~}	{1,~~2~~,~~3~~}	{~~1~~,~~2~~,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	**3**
{1,2,~~3~~}	**3**	{1,2,~~3~~}

Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



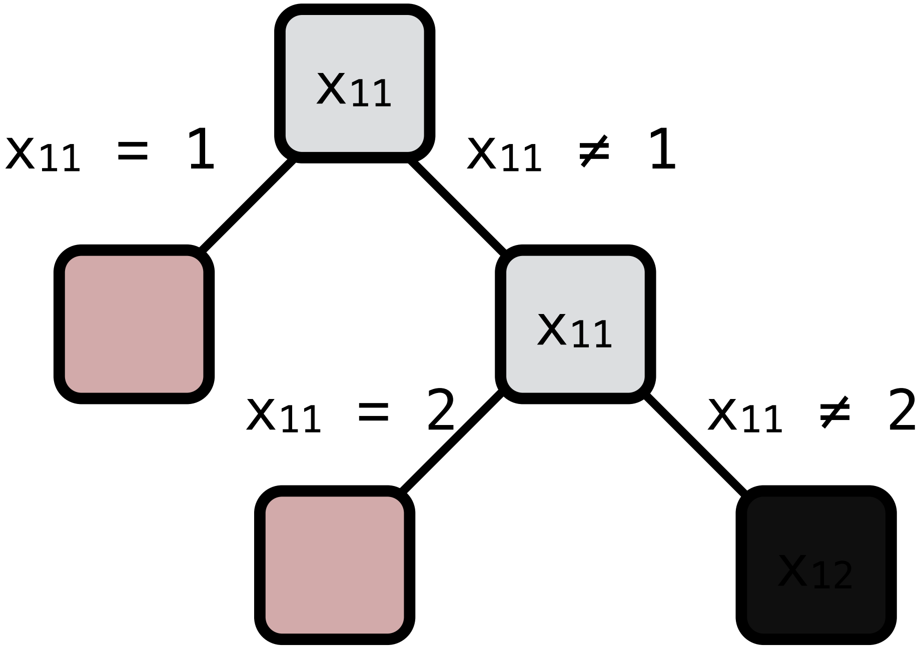


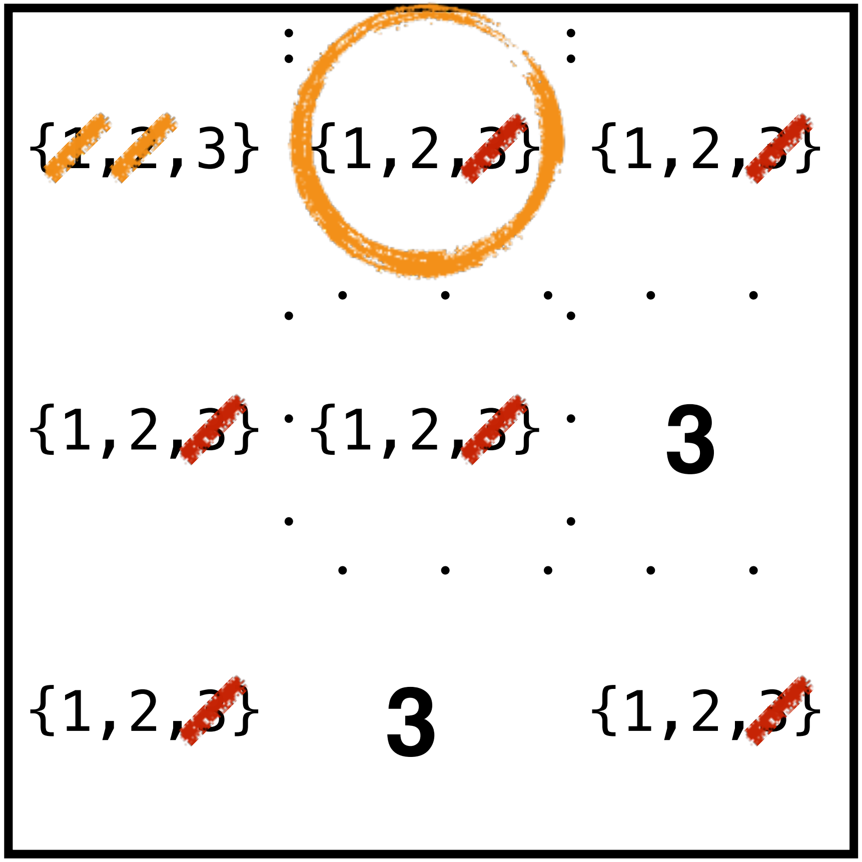
{~~1~~,2,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	**3**
{1,2,~~3~~}	**3**	{1,2,~~3~~}

Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



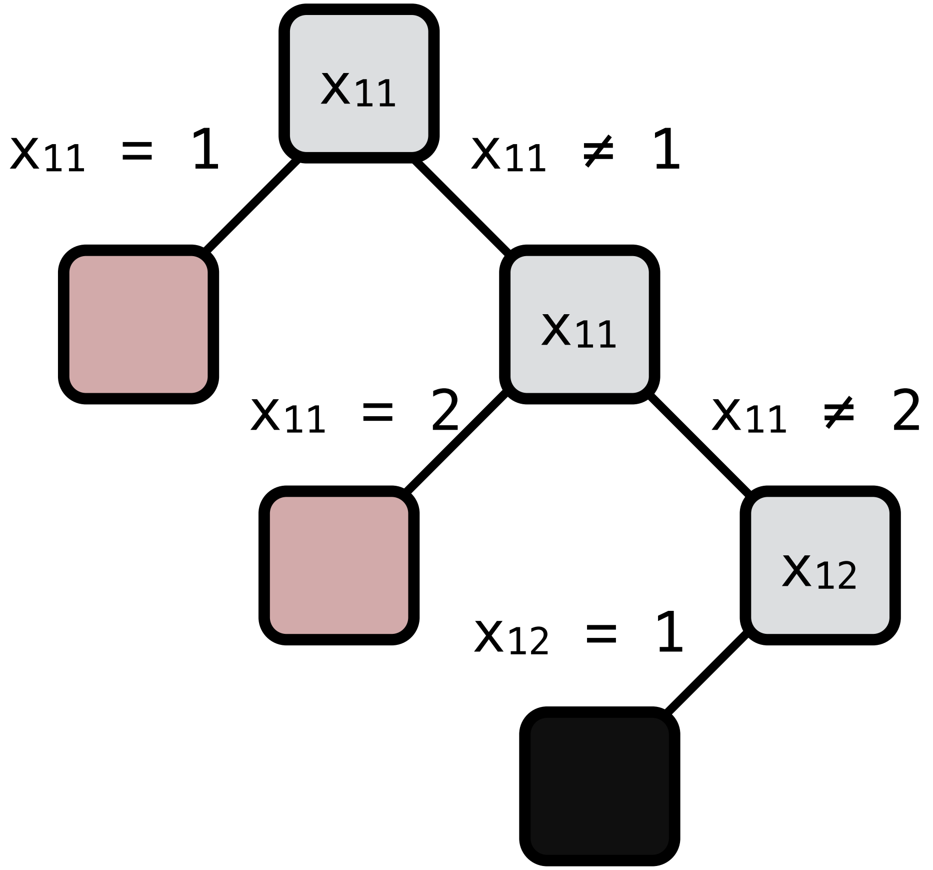


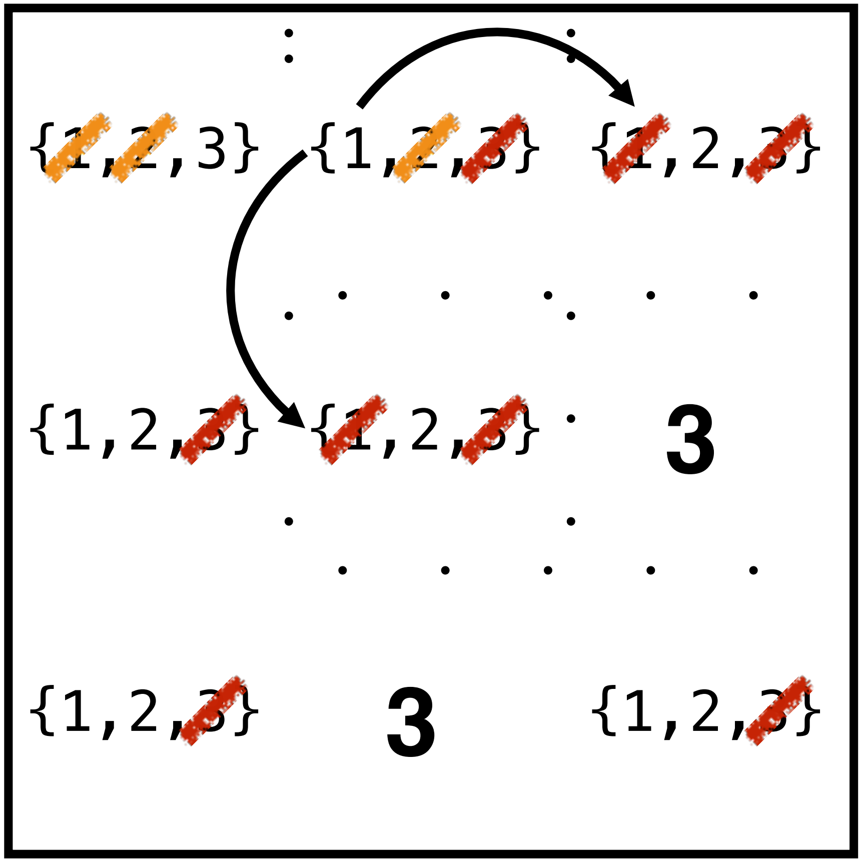
{~~1~~,~~2~~,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored

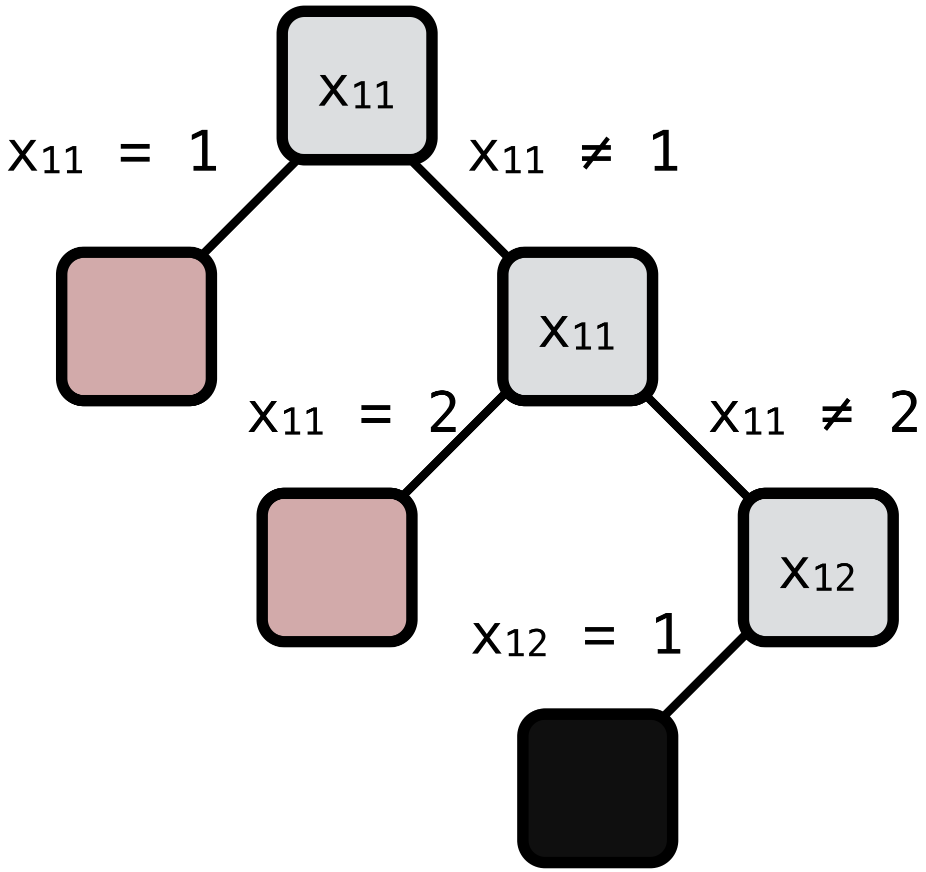


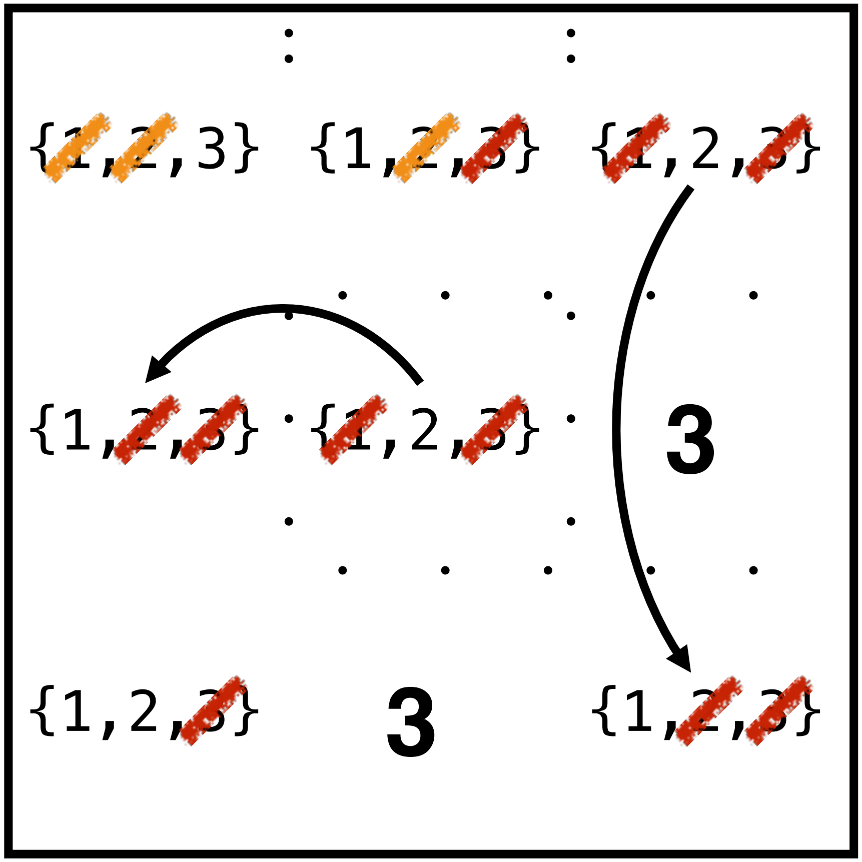


Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored

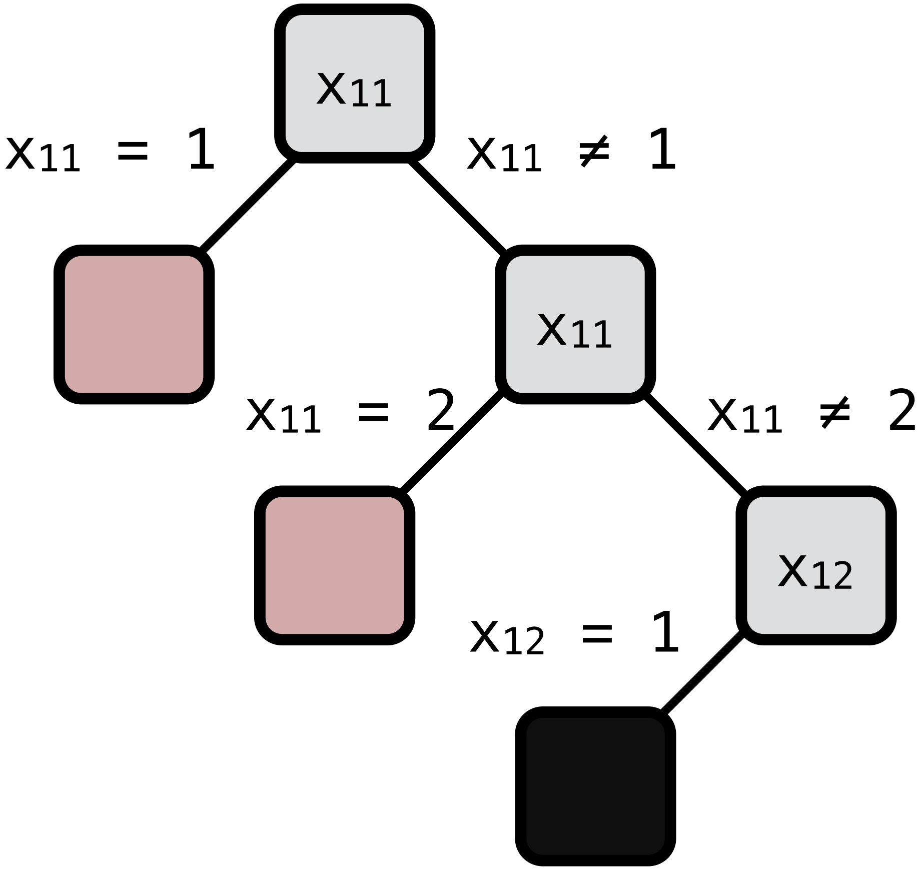


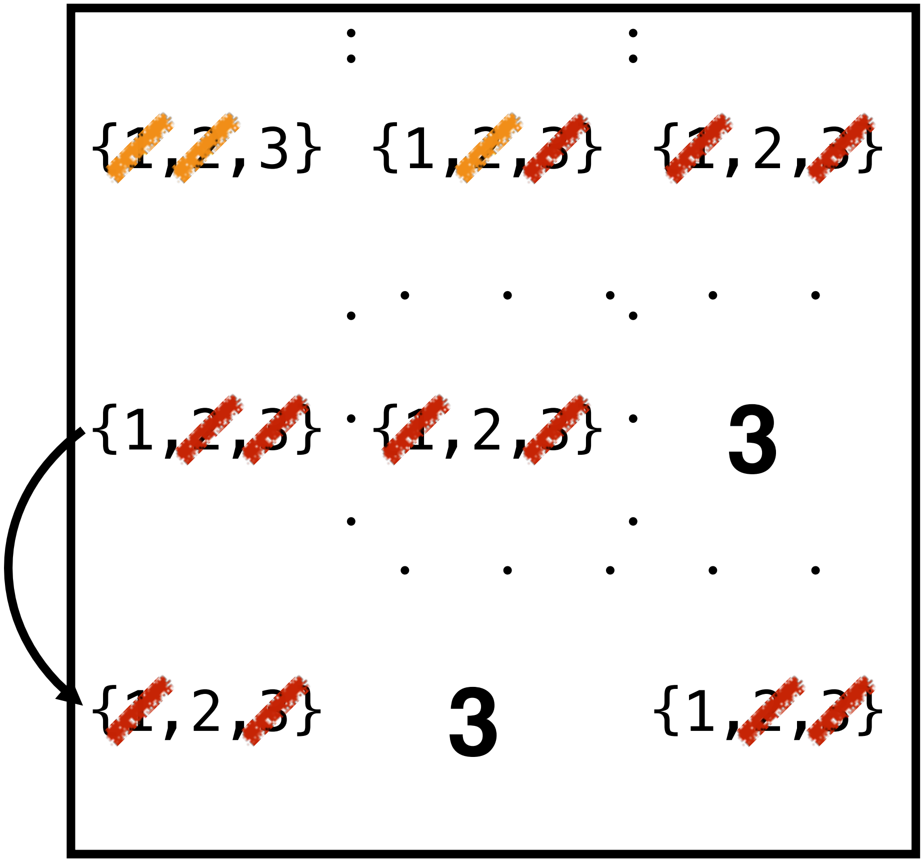


Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored



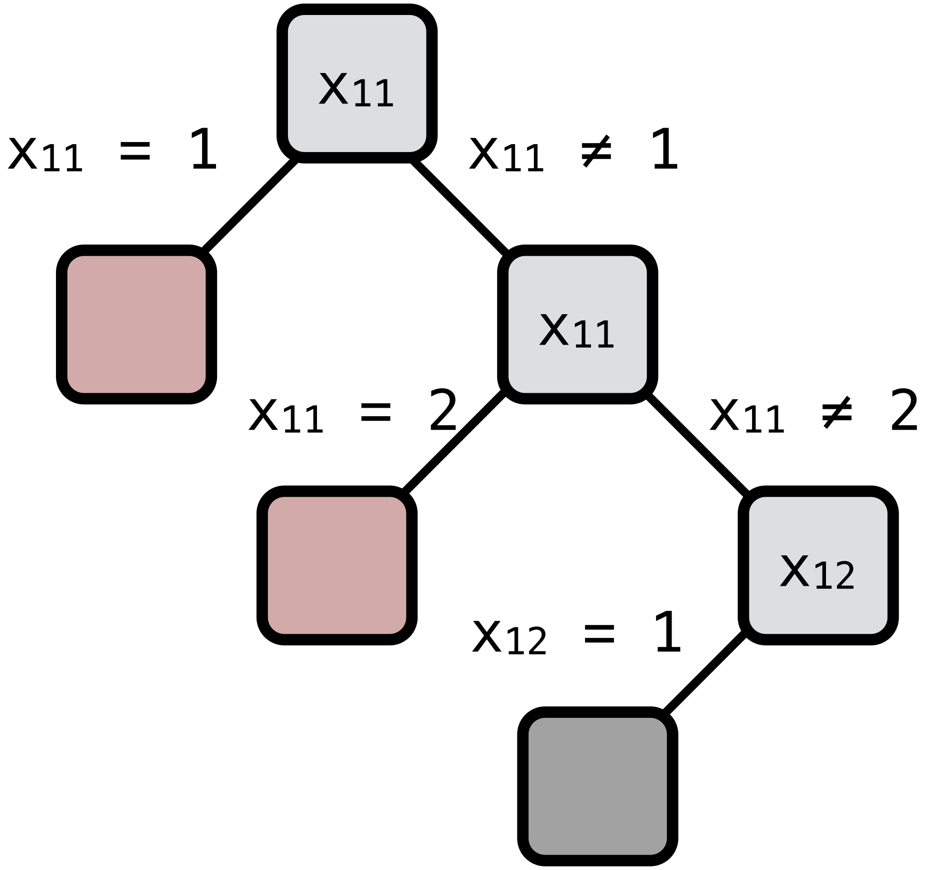


{~~1~~,~~2~~,3}	{1,~~2~~,~~3~~}	{~~1~~,2,~~3~~}
{1,~~2~~,~~3~~}	{~~1~~,2,~~3~~}	3
{~~1~~,2,~~3~~}	3	{1,~~2~~,~~3~~}

Another example

Key mechanism:

- The new constraints narrow the domains
- And cause propagation
- On backtrack, the domains are restored

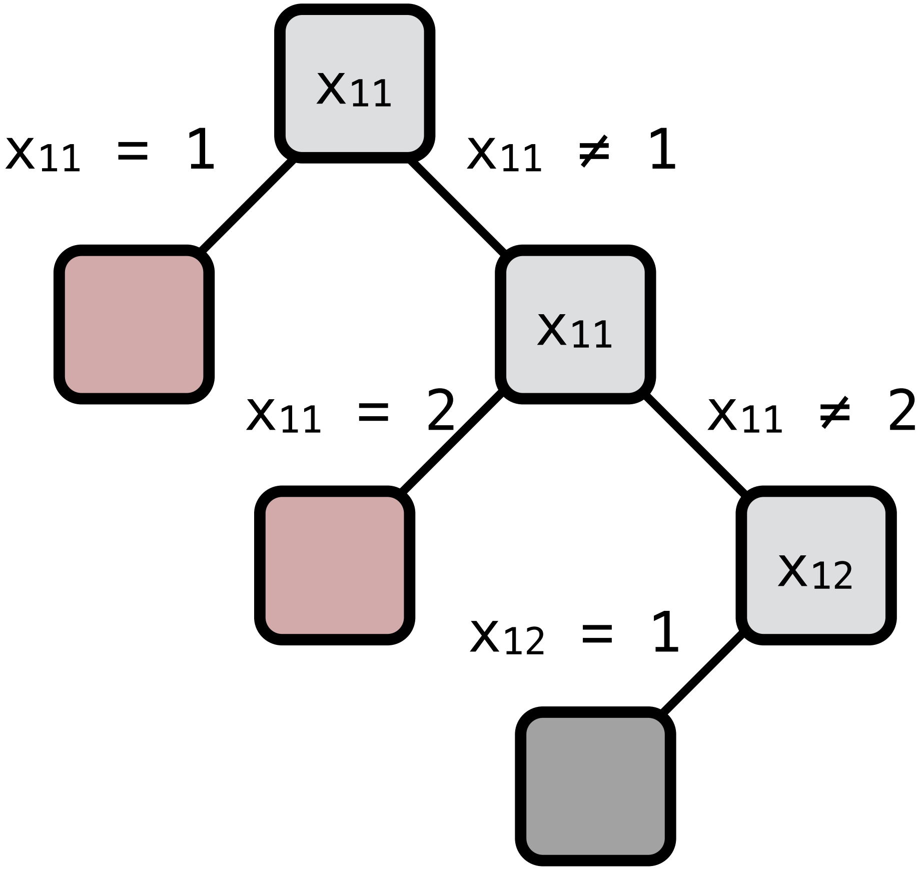


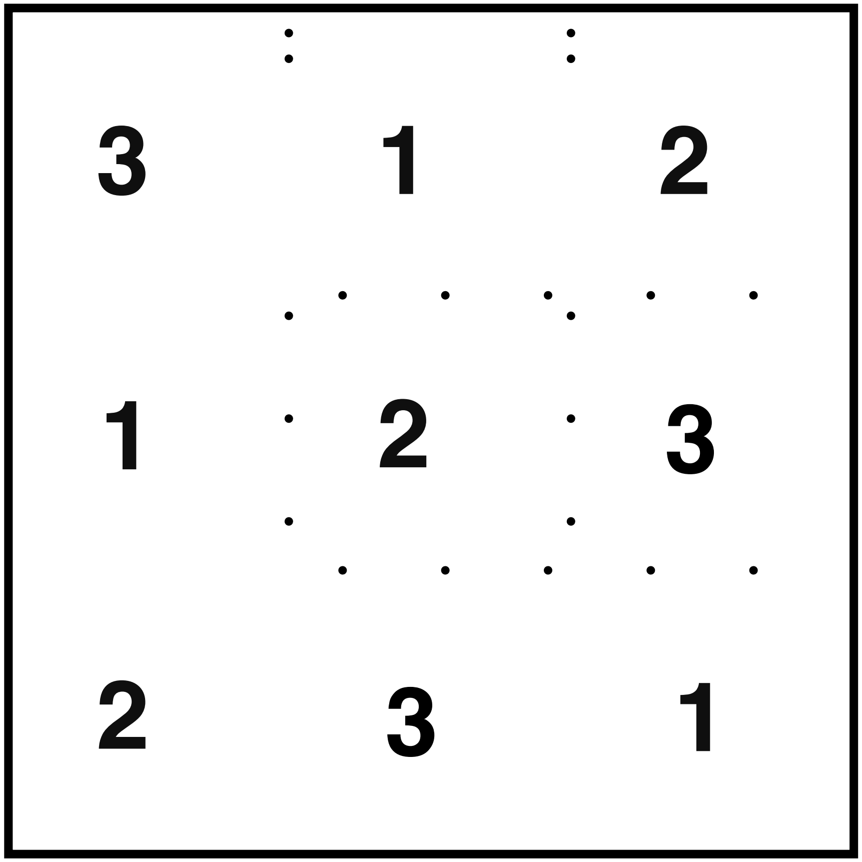
3	1	2
1	2	3
2	3	1

Another example

Propagation can have a huge impact

With: 3 leaves **Without: 2,187 leaves**





3	1	2
1	2	3
2	3	1

Redundant Constraints

Redundant constraints

- Sometimes it is worth adding a constraint
- Even if it is not necessary
- Because of the additional propagation

Example:

- Let's add a redundant constraint

"there must be a 3 in row 1"

$$(x_{11} = 3) \vee (x_{12} = 3) \vee (x_{13} = 3)$$

{1,2,3}	{1,2,3}	{1,2,3}
{1,2,3}	{1,2,3}	**3**
{1,2,3}	**3**	{1,2,3}

Redundant Constraints

Redundant constraints

- Sometimes it is worth adding a constraint
- Even if it is not necessary
- Because of the additional propagation

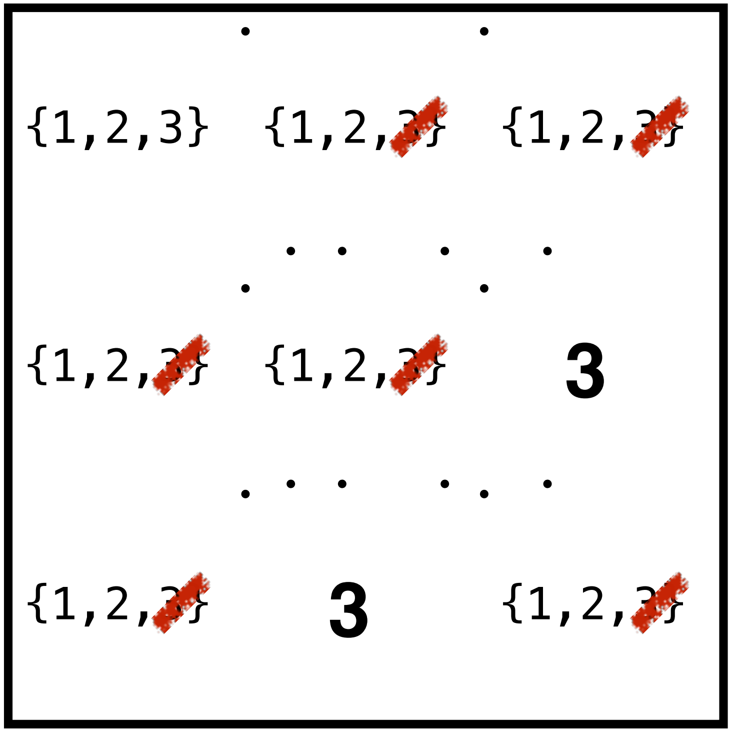
Example:

- Let's add a redundant constraint

"there must be a 3 in row 1"

$$(x_{11} = 3) \vee \underset{\substack{\uparrow \\ cst\ =\ 0}}{(x_{12} = 3)} \vee \underset{\substack{\uparrow \\ cst\ =\ 0}}{(x_{13} = 3)}$$

first round
of propagation



{1,2,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

Redundant Constraints

Redundant constraints

- Sometimes it is worth adding a constraint
- Even if it is not necessary
- Because of the additional propagation

Example:

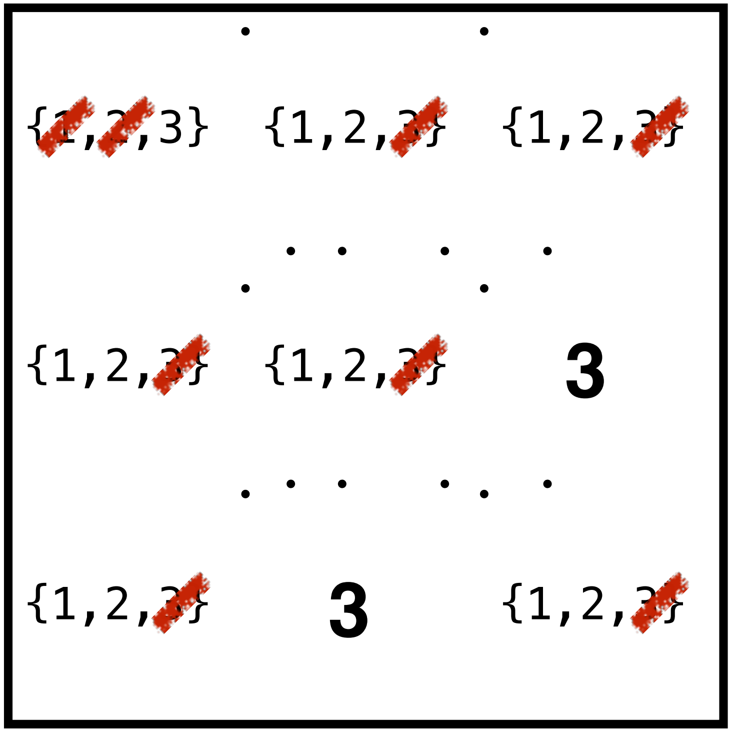
- Let's add a redundant constraint

"there must be a 3 in row 1"

$$\underset{\substack{\downarrow\\ \text{this is 1}}}{(x_{11} = 3)} \vee \underset{\substack{\uparrow\\ \text{cst} = 0}}{(x_{12} = 3)} \vee \underset{\substack{\uparrow\\ \text{cst} = 0}}{(x_{13} = 3)}$$

From here, we find a solution with no fail at all!

first round
of propagation



{~~1~~,~~2~~,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	**3**
{1,2,~~3~~}	**3**	{1,2,~~3~~}

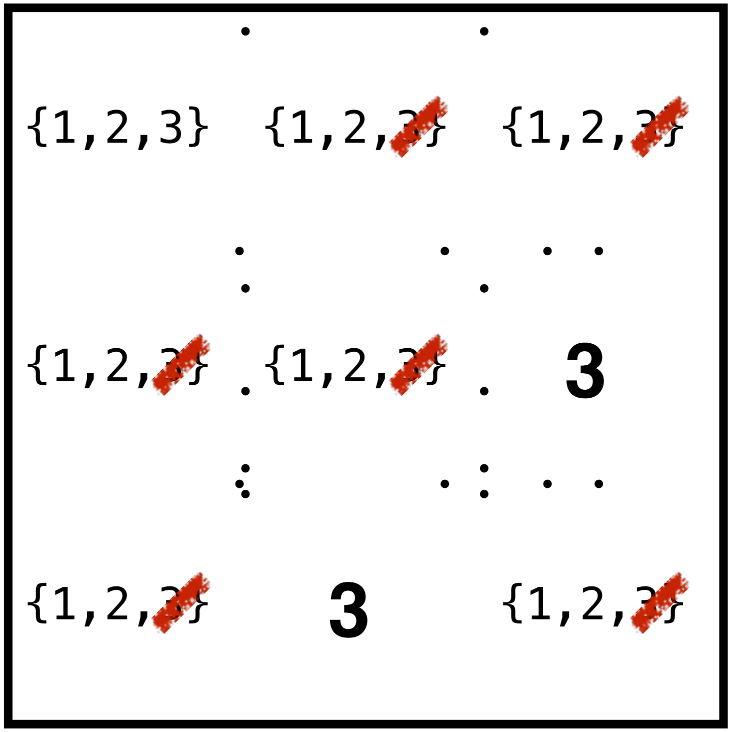
Redundant Constraints

Global Constraints

- A constraint reasoning on many variables at the same time
- Specialized, powerful filtering

Example

- No more propagation after this



{1,2,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

Redundant Constraints

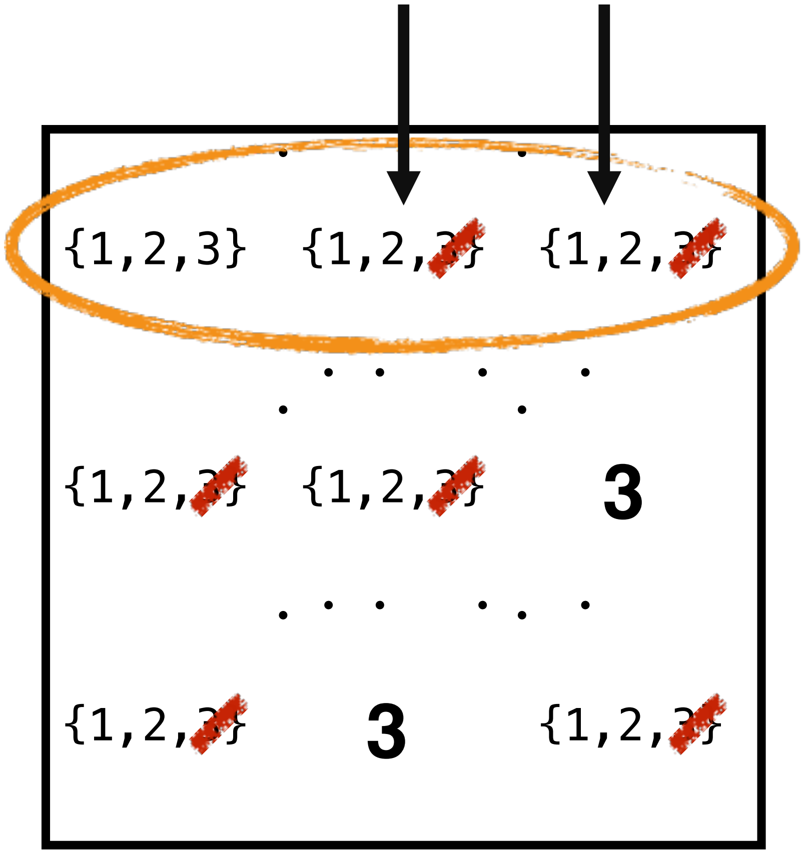
Global Constraints

- A constraint reasoning on many variables at the same time
- Specialized, powerful filtering

Example

- No more propagation after this
- But if we reason on a whole row…

values 1,2 must go to those vars



{1,2,3} {1,2,~~3~~} {1,2,~~3~~}

{1,2,~~3~~} {1,2,~~3~~} 3

{1,2,~~3~~} 3 {1,2,~~3~~}

Redundant Constraints

Global Constraints

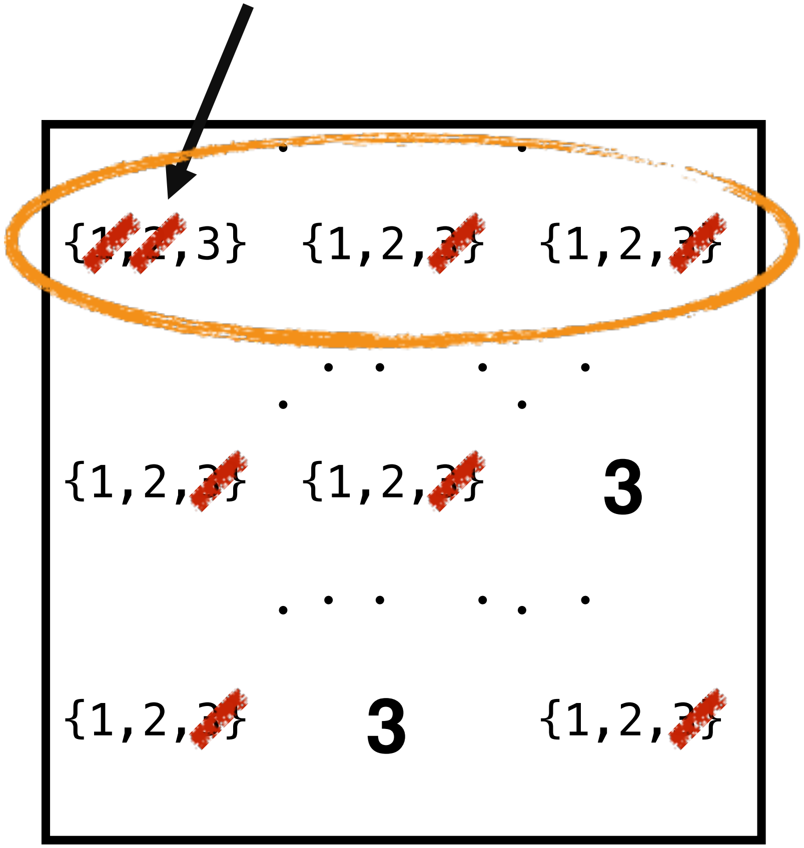
- A constraint reasoning on many variables at the same time
- Specialized, powerful filtering

Example

- No more propagation after this
- But if we reason on a whole row…
- …we can deduce (and filter) more

Remember: from here, we find a solution with no fail at all!

so, this must be 3



{~~1~~,~~2~~,3}	{1,2,~~3~~}	{1,2,~~3~~}
{1,2,~~3~~}	{1,2,~~3~~}	3
{1,2,~~3~~}	3	{1,2,~~3~~}

Redundant Constraints

Meta constraints vs globals

- Meta-constraints allow to model just about everything
- But they often have poor filtering
- Advice: use globals whenever it is possible

Redundant constraints vs globals

- Redundant constraints must be carefully engineered based on domain knowledge
- But they provide some“global” propagation
- Advice: add if the additional propagation is not subsumed