Chapter 5: Global Constraints(Sudoku)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Problem
Program
Initial Propagation (Forward Checking)
Improved Reasoning
Search

Licence

Outline

What we want to introduce

- Global Constraints
  - Powerful modelling abstractions
  - Non-trivial propagation
- Consistency Levels
  - Tradeoff between speed and propagation
  - Characterisation of reasoning power
- Example: Alldifferent
  - 3 variants shown

Methodology

- Evaluation on Sudoku puzzle
- Comparing
  - Initial setup
  - Search
  - Performance
- Explaining reasoning inside constraint
- Link to general classification of global constraints

Problem Definition

Sudoku

Fill in numbers from 1 to 9 so that each row, column and block contain each number exactly once

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

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

Model

- A variable for each cell, ranging from 1 to 9
- A 9x9 matrix of variables describing the problem
- Preassigned integers for the given hints
- `alldifferent` constraints for each row, column and 3x3 block

Reminder: `alldifferent`

- Argument: list of variables
- Meaning: variables are pairwise different
- Reasoning: Forward Checking (FC)
  - When variable is assigned to value, remove the value from all other variables
  - If a variable has only one possible value, then it is assigned
  - If a variable has no possible values, then the constraint fails
  - Constraint is checked whenever one of its variables is assigned
  - Equivalent to decomposition into binary disequality constraints

Declarations

```
:-module(sudoku).
:-export(top/0).
:-lib(ic).

top:-
    problem(Matrix),
    model(Matrix),
    writeln(Matrix).
```


Data

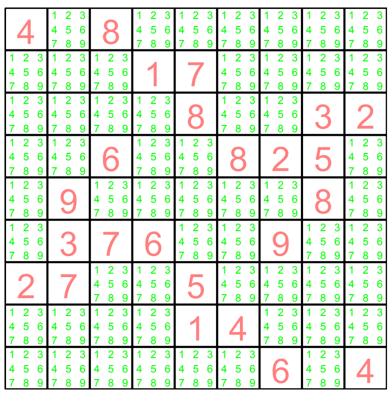
```
problem([]([](4, _, 8, _, _, _, _, _, _),
           [](_, _, _, 1, 7, _, _, _, _),
           [](_, _, _, _, 8, _, _, 3, 2),
           [](_, _, 6, _, _, 8, 2, 5, _),
           [](_, 9, _, _, _, _, _, 8, _),
           [](_, 3, 7, 6, _, _, 9, _, _),
           [](2, 7, _, _, 5, _, _, _, _),
           [](_, _, _, _, 1, 4, _, _, _),
           [](_, _, _, _, _, _, 6, _, 4))).
```

Main Program

```
model(Matrix):-
    Matrix[1..9,1..9] :: 1..9,
    (for(I,1,9),
     param(Matrix) do
        alldifferent(Matrix[I,1..9]),
        alldifferent(Matrix[1..9,I])
    ),
    (multifor([I,J],[1,1],[7,7],[3,3]),
     param(Matrix) do
        alldifferent(flatten(Matrix[I..I+2,J..J+2]))
    ),
    flatten_array(Matrix,List),
    labeling(List).
```

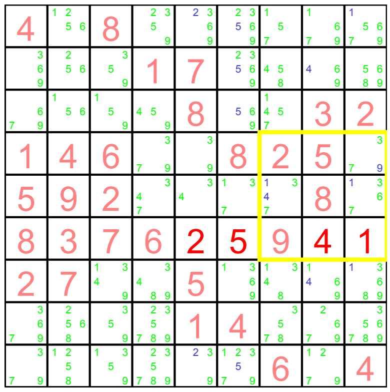

Domain Visualizer

- Problem shown as matrix
- Each cell corresponds to a variable
- Instantiated: Shows integer value (large)
- Uninstantiated: Shows values in domain

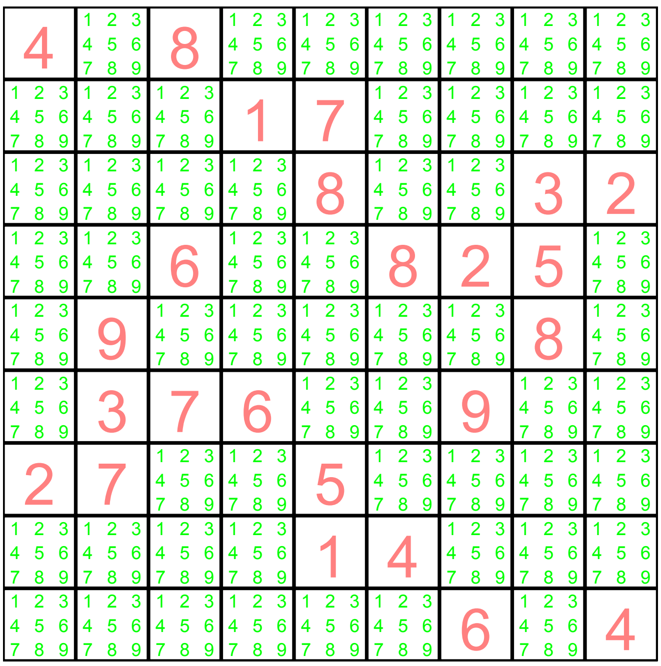


Constraint Visualizer

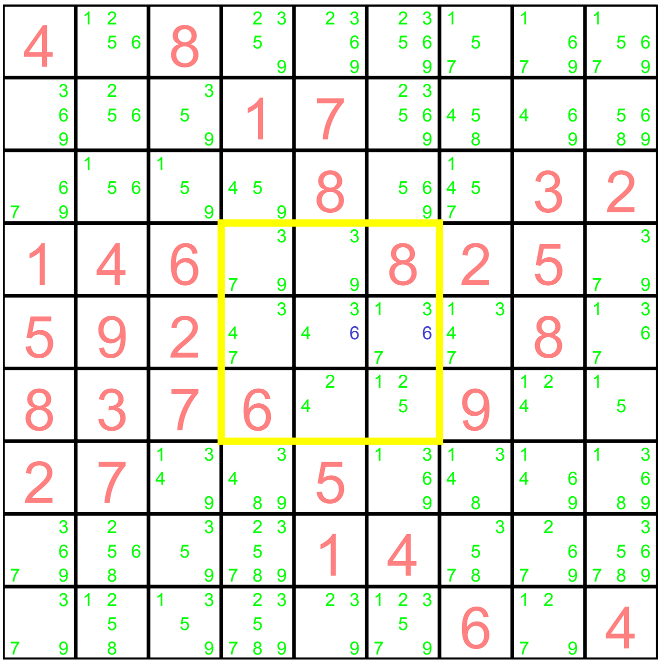
- Problem shown as matrix
- Currently active constraint highlighted
- Values removed at this step shown in blue
- Values assigned at this step shown in red



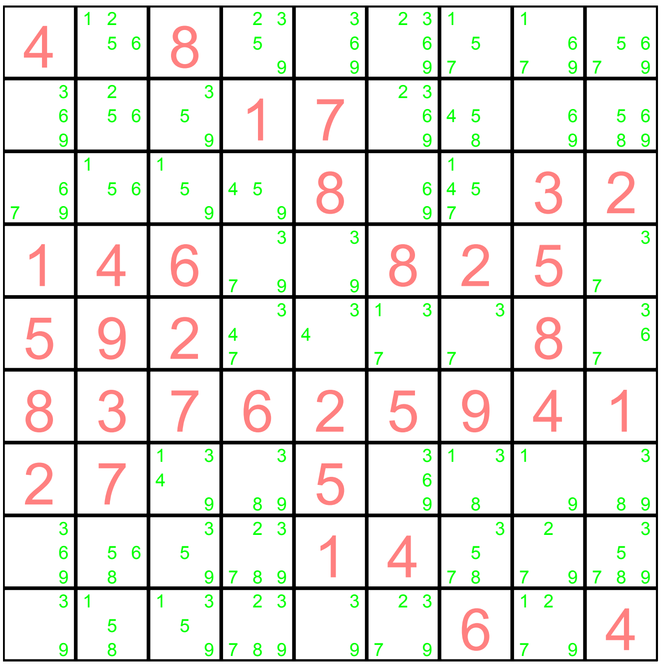
Initial State (Forward Checking)



Propagation Steps (Forward Checking)



After Setup (Forward Checking)



Can we do better?

- The alldifferent constraint is missing propagation
  - How can we do more propagation?
  - Do we know when we derive all possible information from the constraint?
- Constraints only interact by changing domains of variables

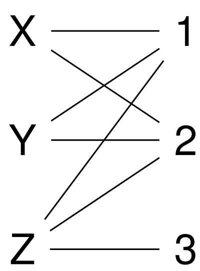
A Simpler Example

```
:-lib(ic).

top:-
    X :: 1..2,
    Y :: 1..2,
    Z :: 1..3,
    alldifferent([X,Y,Z]),
    writeln([X,Y,Z]).
```

Using Forward Checking

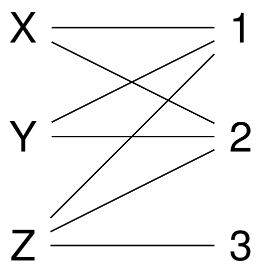
- No variable is assigned
- No reduction of domains
- But, values 1 and 2 can be removed from Z
- This means that Z is assigned to 3

Visualization of alldifferent as Graph



- Show problem as graph with two types of nodes
  - Variables on the left
  - Values on the right
- If value is in domain of variable, show link between them
- This is called a *bipartite* graph

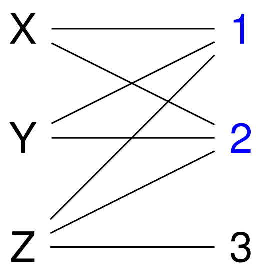
A Simpler Example



Value Graph for

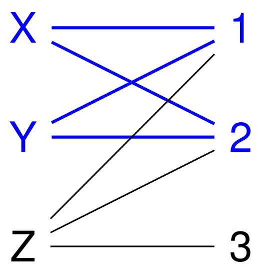
```
X :: 1..2,

Y :: 1..2,

Z :: 1..3
```


A Simpler Example



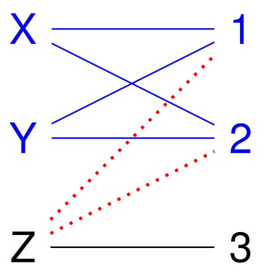
Check interval [1,2]

A Simpler Example



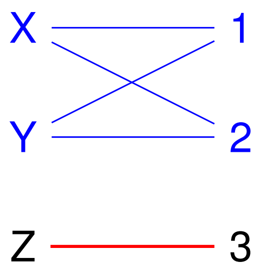
- Find variables completely contained in interval
- There are two: X and Y
- This uses up the capacity of the interval

A Simpler Example



No other variable can use that interval

A Simpler Example



Only one value left in domain of Z, this can be assigned

Idea (Hall Intervals)

- Take each interval of possible values, say size N
- Find all K variables whose domain is completely contained in interval
- If $K > N$ then the constraint is infeasible
- If $K = N$ then no other variable can use that interval
- Remove values from such variables if their bounds change
- If $K < N$ do nothing
- Re-check whenever domain bounds change

Implementation

- Problem: Too many intervals ($O(n^2)$) to consider
- Solution:
  - Check only those intervals which update bounds
  - Enumerate intervals incrementally
  - Starting from lowest(highest) value
  - Using sorted list of variables
- Complexity: $O(n \log(n))$ in standard implementations
- Important: Only looks at min/max bounds of variables

Bounds Consistency

Definition

A constraint achieves *bounds consistency*, if for the lower and upper bound of every variable, it is possible to find values for all other variables between their lower and upper bounds which satisfy the constraint.

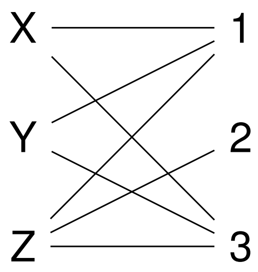
Can we do better?

- Bounds consistency only considers min/max bounds
- Ignores “holes” in domain
- Sometimes we can improve propagation looking at those holes

Another Simple Example

```
:-lib(ic).

top:-
    X :: [1,3],
    Y :: [1,3],
    Z :: 1..3,
    alldifferent([X,Y,Z]),
    writeln([X,Y,Z]).
```

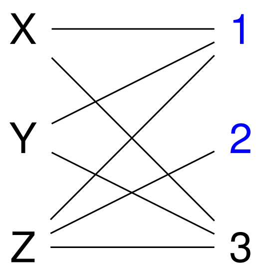
Another Simple Example



Value Graph for

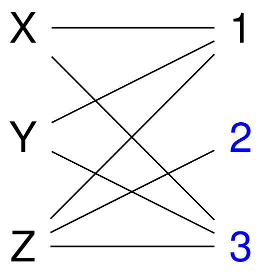
```
X :: [1,3],
Y :: [1,3],
Z :: 1..3
```


Another Simple Example



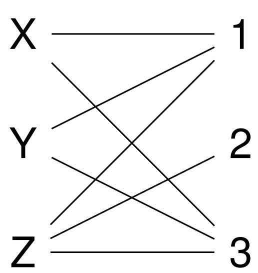
- Check interval [1,2]
- No domain of a variable completely contained in interval
- No propagation

Another Simple Example



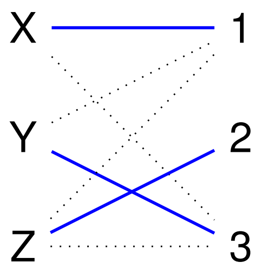
- Check interval [2,3]
- No domain of a variable completely contained in interval
- No propagation

Another Simple Example



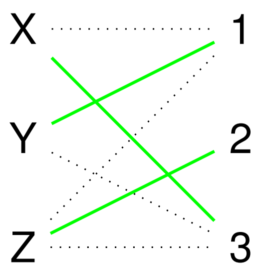
But, more propagation is possible, there are only two solutions

Another Simple Example



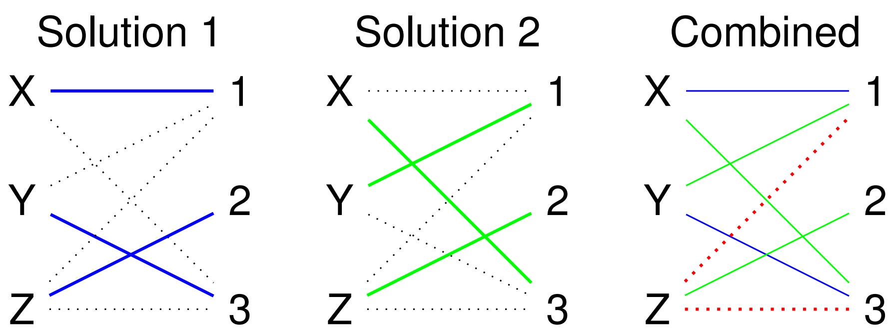
Solution 1: assignment in blue

Another Simple Example



Solution 2: assignment in green

Another Simple Example



Combining solutions shows that Z=1 and Z=3 are not possible. Can we deduce this without enumerating solutions?

Solutions and maximal matchings

- A *Matching* is subset of edges which do not coincide in any node
- No matching can have more edges than number of variables
- Every solution corresponds to a *maximal matching* and vice versa
- If a link does not belong to some maximal matching, then it can be removed

Implementation

- Possible to compute all links which belong to some matching
  - Without enumerating all of them!
- Enough to compute **one** maximal matching
- Requires algorithm for *strongly connected components*
- Extra work required if more values than variables
- All links (values in domains) which are not supported can be removed
- Complexity: $O(n^{1.5}d)$

Domain Consistency

Definition

A constraint achieves *domain consistency*, if for every variable and for every value in its domain, it is possible to find values in the domains of all other variables which satisfy the constraint.

- Also called *generalized arc consistency (GAC)*
- or *hyper arc consistency*

Can we still do better?

- NO! This extracts all information from this one constraint
- We could perhaps improve speed, but not propagation
- But possible to use different model
- Or model interaction of multiple constraints

Should all constraints achieve domain consistency?

- Domain consistency is usually more expensive than bounds consistency
  - Overkill for simple problems
  - Nice to have choices
- For some constraints achieving domain consistency is NP-hard
  - We have to live with more restricted propagation

Improved Propagation in ECLiPSe

- `ic_global` library bounds consistent version
- `ic_global_gac` library domain consistent version
- Choose which version to use by using module annotation
- Choice can be passed as parameter

Declarations

```
:-module(sudoku).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).
:-lib(ic_global_gac).

top:-
    problem(Matrix),
    model(ic_global,Matrix),
    writeln(Matrix).
```

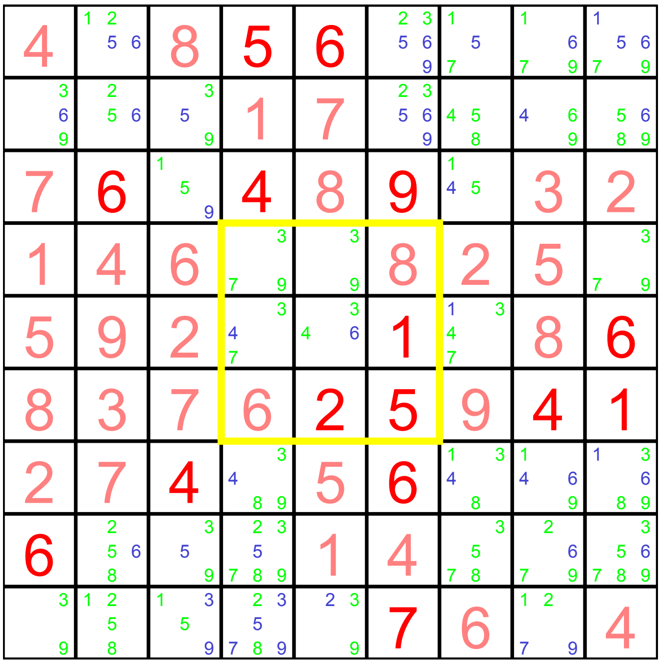
Main Program

```
model(Method,Matrix):-
    Matrix[1..9,1..9] :: 1..9,
    (for(I,1,9),
     param(Method,Matrix) do
        Method:alldifferent(Matrix[I,1..9]),
        Method:alldifferent(Matrix[1..9,I])
    ),
    (multifor([I,J],[1,1],[7,7],[3,3]),
     param(Method,Matrix) do
        Method:alldifferent(flatten(Matrix[I..I+2,
                                           J..J+2]))
    ),
    flatten_array(Matrix,List),labeling(List).
```

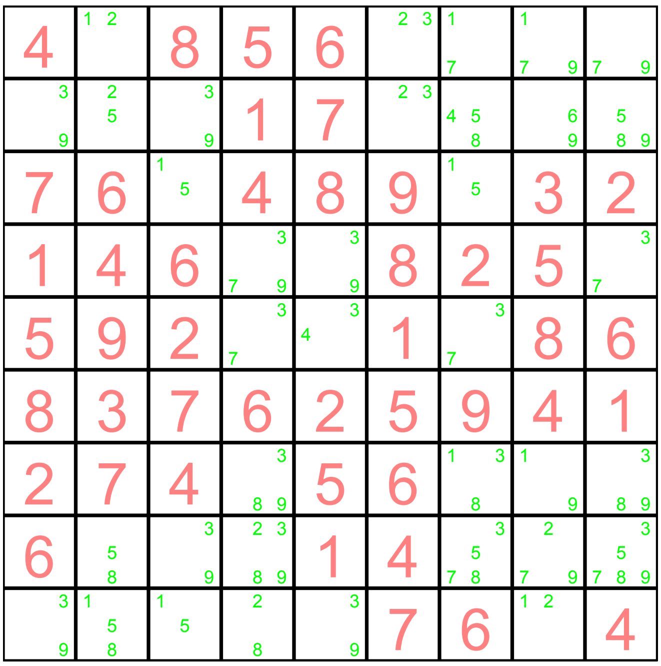

Initial State (Bounds Consistency)

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

Propagation Steps (Bounds Consistency)



After Setup (Bounds Consistency)



Initial State (Domain Consistency)

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

Propagation Steps (Domain Consistency)

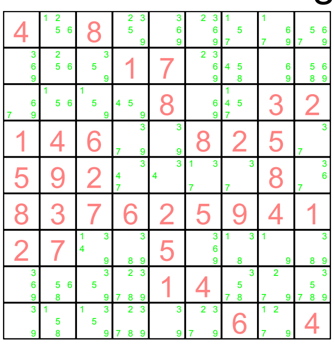
4	2	8	5	6	3	1	1 6 7 9	1 5 6 7 9
3 6 9	5	3 5 9	1	7	2	4	6	8
7	6	1	4	8	9	5	3	2
1	4	6	3 7 9	3 9	8	2	5	3 7 9
5	9	2	3 7	4	1	1 3 4 7	8	6
8	3	7	6	2	5	9	4	1
2	7	4	3 8 9	5	6	8	1	1 3 6 8 9
6	8	3 5 9	2	1	4	3 5 7 8	2 6 7 9	5
3 9	1	5	8	2 3 9	7	6	2	4

After Setup (Domain Consistency)

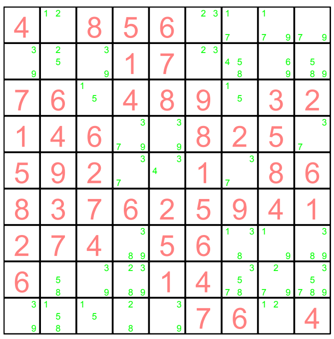
4	2	8	5	6	3	1	7 9	7 9
3 9	5	3 9	1	7	2	4	6	8
7	6	1	4	8	9	5	3	2
1	4	6	3 7 9	3 9	8	2	5	3 7
5	9	2	3 7	4	1	3 7	8	6
8	3	7	6	2	5	9	4	1
2	7	4	3 9	5	6	8	1	3 9
6	8	3 9	2	1	4	3 7	7 9	5
3 9	1	5	8	3 9	7	6	2	4

Comparison

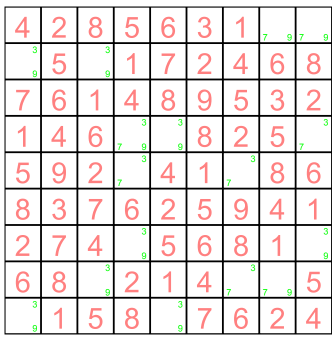
Forward Checking



Bounds Consistency



Domain Consistency



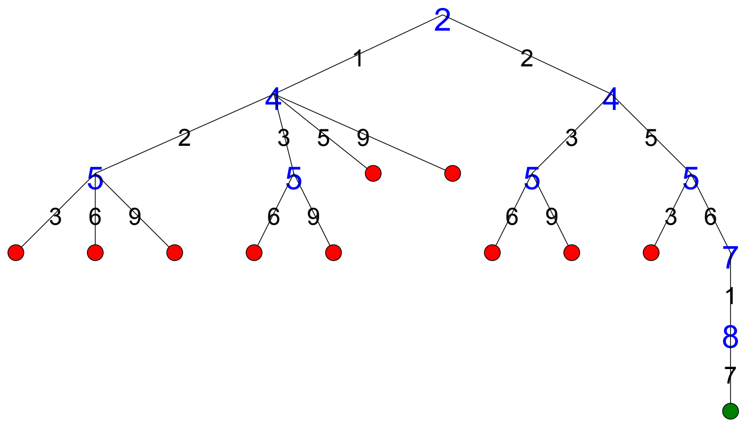
Typical?

- This does not always happen
- Sometimes, two methods produce same amount of propagation
- Possible to predict in certain special cases
- In general, tradeoff between speed and propagation
- Not always fastest to remove inconsistent values early
- But often required to find a solution at all

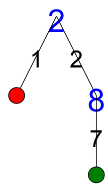
Simple search routine

- Enumerate variables in given order
- Try values starting from smallest one in domain
- Complete, chronological backtracking

Search Tree (Forward Checking)



Search Tree (Bounds Consistency)



Search Tree (Domain Consistency)

Observations

- Search tree much smaller for bounds/domain consistency
- Does not always happen like this
- Smaller tree = Less execution time
- Less reasoning = Less execution time
- Problem: Finding best balance
- For Sudoku: not good enough, should not require any search!

Solution

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

Global Constraints

- Powerful modelling abstractions
- Efficient reasoning

Consistency Levels

- Defined levels of propagation
- Tradeoff speed/reasoning
- Characterisation of power of constraint

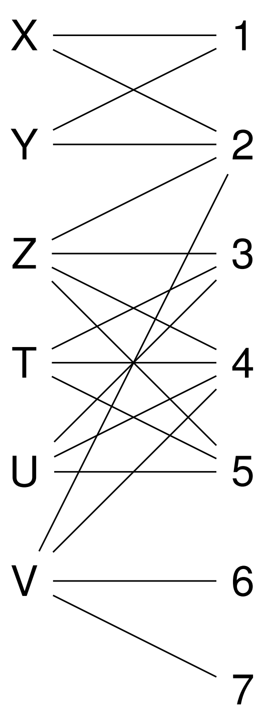
Alldifferent Variants

- Forward Checking
  - Only reacts when variables are assigned
  - Equivalent to decomposition into binary constraints
- Bounds Consistency
  - Typical best compomise speed/reasoning
  - Works well if no holes in domain
- Domain Consistency
  - Extracts all information from single constraint
  - Cost only justified for very hard problems

Bigger Example

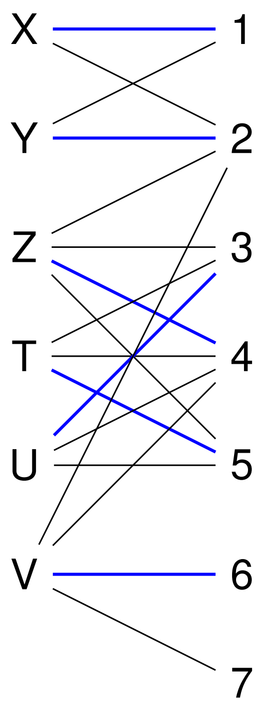
```
:-lib(ic).
:-lib(ic_global_gac).

top:-
    [X,Y] :: 1..2,
    Z :: 2..5,
    [T,U] :: 3..5,
    V :: [2,4,6,7],
    ic_global_gac:alldifferent([X,Y,Z,T,U,V]).
```


Making constraint domain consistent



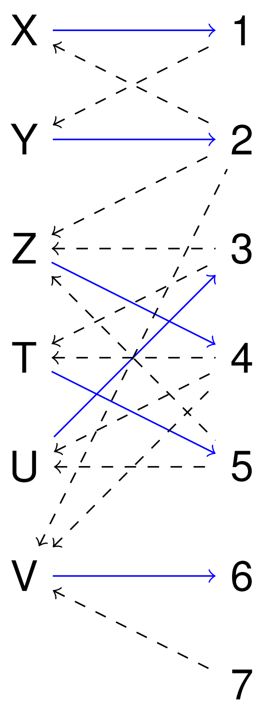
Problem shown as bipartite graph

Making constraint domain consistent



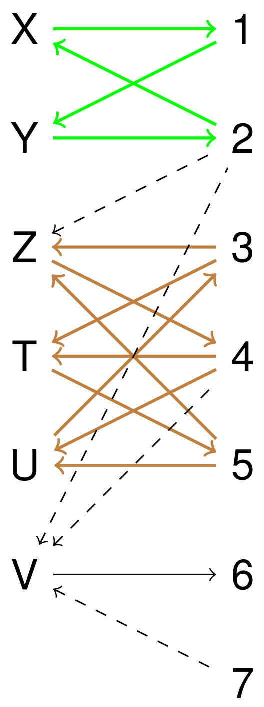
Find maximal matching (in blue)

Making constraint domain consistent



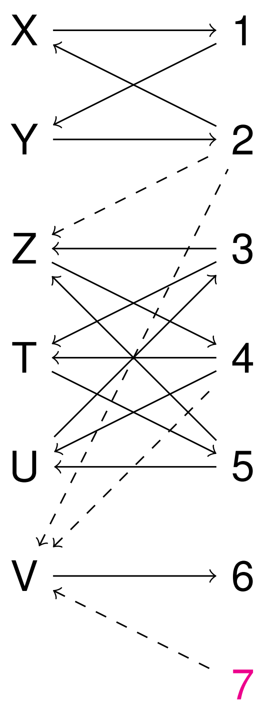
Orient graph (edges in matching from variables to values, all others from values to variables), mark edges in matching

Making constraint domain consistent



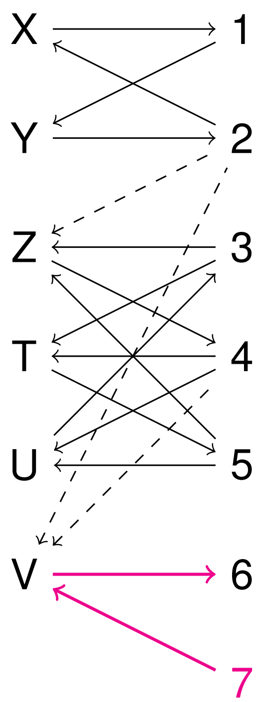
Find strongly connected components (green and brown), mark their edges

Making constraint domain consistent



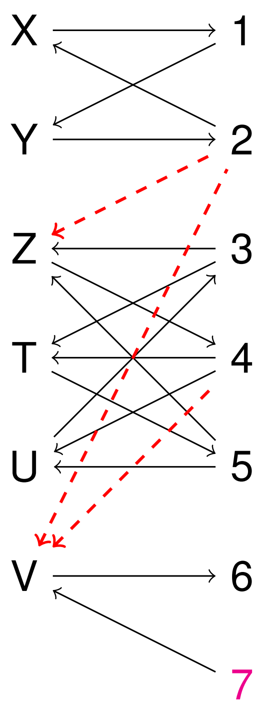
Find unmatched value nodes (here node 7, magenta)

Making constraint domain consistent



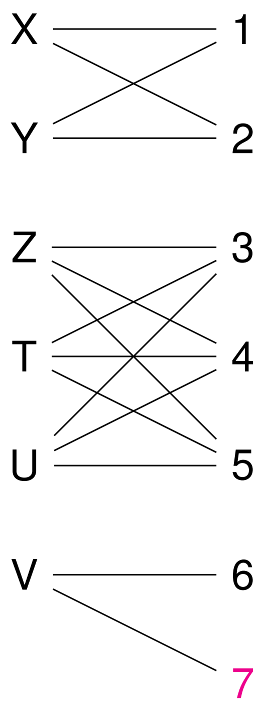
Find alternating paths from such nodes (in magenta), mark their edges

Making constraint domain consistent



All unmarked edges can be removed

Making constraint domain consistent

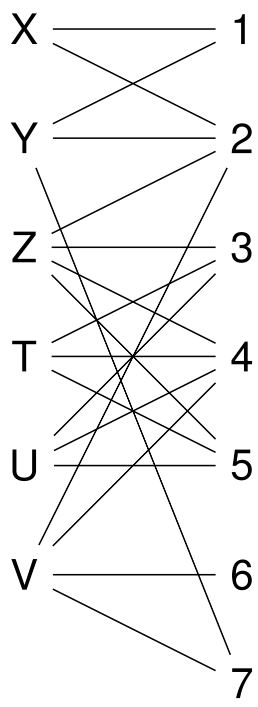


Resulting graph, constraint is domain consistent

Extended Example

```
:-lib(ic).
:-lib(ic_global_gac).

top:-
    X :: 1..2,
    Y :: [1,2,7],
    Z :: 2..5,
    [T,U] :: 3..5,
    V :: [2,4,6,7],
    ic_global_gac:alldifferent([X,Y,Z,T,U,V]).
```

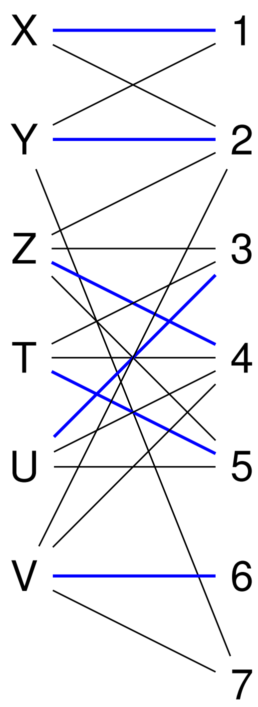
No propagation in expanded example



Problem shown as bipartite graph

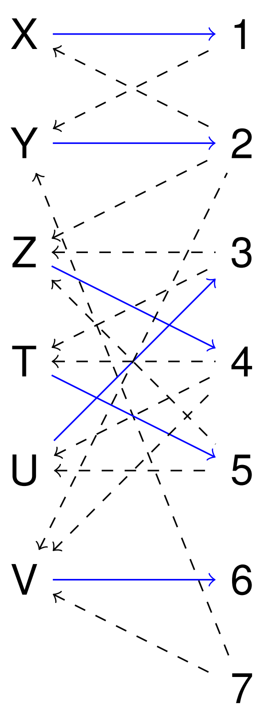
Conclusions
Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

No propagation in expanded example



Find maximal matching (in blue)

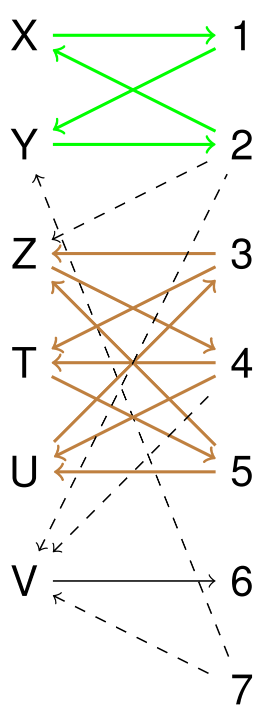
No propagation in expanded example



Orient graph (edges in matching from variables to values, all others from values to variables), mark edges in matching

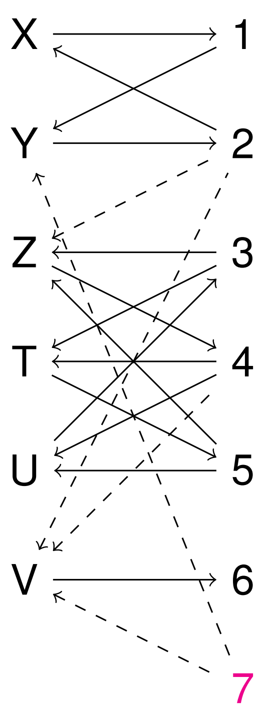
Conclusions
Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

No propagation in expanded example



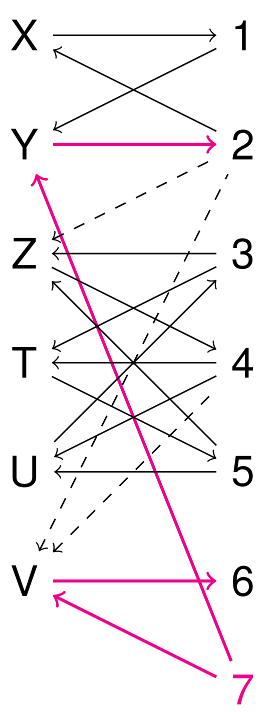
Find strongly connected components (green and brown), mark their edges

No propagation in expanded example



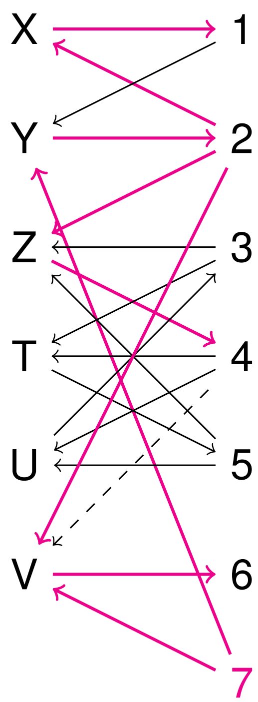
Find unmatched value nodes (here node 7, magenta)

No propagation in expanded example



Find alternating paths from such nodes (in magenta), mark their edges

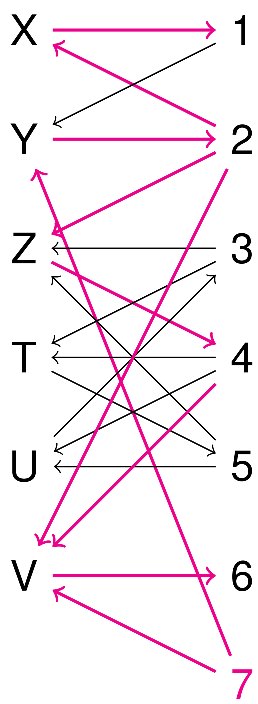
No propagation in expanded example



Continue with alternating paths

Conclusions
Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

No propagation in expanded example



Continue with alternating paths, all edges marked, no propagation, constraint is domain consistent

Observation

- A lot of effort for no propagation
- Problem: Slows down search without any upside
- Constraint is woken every time any domain is changed
- How often does the constraint do actual pruning?

Generalize Program for different sizes

- How to generalize program for different sizes (4,9,16,25,36...)
- Add parameter R (Order, number of blocks in a row/column)
- Size N is square of R
- Remove explicit integer bounds by expressions
- Useful to do this change as rewriting of working program

Main Program

```
model(R,M,Matrix):-
    N is R*R,Matrix[1..N,1..N] :: 1..N,
    (for(I,1,N),
     param(N,M,Matrix) do
        M:alldifferent(Matrix[I,1..N]),
        M:alldifferent(Matrix[1..N,I])
    ),
    (multifor([I,J],[1,1],[N-R+1,N-R+1],[R,R]),
     param(R,M,Matrix) do
        M:alldifferent(flatten(Matrix[I..I+R-1,
                                      J..J+R-1]))
    ),
    flatten_array(Matrix,List),labeling(List).
```


Conclusions
Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

More Information

Krzysztof R. Apt and Mark Wallace.
Constraint Logic Programming using ECLiPSe.
Cambridge University Press, New York, NY, USA, 2007.

Willem Jan van Hoeve.
The alldifferent constraint: A survey.
CoRR, cs.PL/0105015, 2001.

Claude-Guy Quimper.
Efficient Propagators for Global Constraints.
PhD thesis, University of Waterloo, 2006.

More Information

H. Simonis.
Sudoku as a constraint problem.
In B. Hnich, P. Prosser, and B. Smith, editors, *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 13–27, September 2005.

I. Lynce and J. Ouaknine.
Sudoku as a SAT problem.
In *9th International Symposium on Artificial Intelligence and Mathematics*, January 2006.

Conclusions
Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

More Information

H. Simonis.
Kakuro as a constraint problem.
In P. Flener, H. Simonis, editors, *Proceedings of the 8th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, September 2008.

Exercises

1.