Constraint Programming

Introduction (www.minicp.org)
(version of 31 October 2022)
“Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.” (E. Freuder)
State Problem = Declarative Programming

Declarative programming is a *programming paradigm* that expresses the logic of a computation without describing its control flow.

Declarative programming for solving constrained combinatorial (optimization) problems means that you express the properties of solutions that must be found by “the solver”.
Counterexample of Declarative Programming

```python
# A Python algorithm for solving a Sudoku

def findNextCellToFill(grid, i, j):
    for x in range(i, 9):
        for y in range(j, 9):
            if grid[x][y] == 0:
                return x, y
    for x in range(0, 9):
        for y in range(0, 9):
            if grid[x][y] == 0:
                return x, y
    return -1, -1

def isValid(grid, i, j, e):
    rowOk = all([e != grid[i][x] for x in range(9)])
    if rowOk:
        columnOk = all([e != grid[x][j] for x in range(9)])
        if columnOk:
            # finding the top left x,y co-ordinates of the section containing the i,j cell
            secTopX, secTopY = 3 * (i // 3), 3 * (j // 3)  # floored quotient should be used here.
            for x in range(secTopX, secTopX + 3):
                for y in range(secTopY, secTopY + 3):
                    if grid[x][y] == e:
                        return False
            return True
    return False

def solveSudoku(grid, i=0, j=0):
    i, j = findNextCellToFill(grid, i, j)
    if i == -1:
        return True
    for e in range(1, 10):
        if isValid(grid, i, j, e):
            grid[i][j] = e
            if solveSudoku(grid, i, j):
                return True
            # Undo the current cell for backtracking
            grid[i][j] = 0
    return False
```

A Python algorithm for solving a Sudoku
What We Mean by Declarative Programming

```
from Numberjack import *

def get_model(N, clues):
    grid = Matrix(N*N, N*N, 1, N*N)
    sudoku = Model(
        [AllDiff(row) for row in grid.row],
        [AllDiff(col) for col in grid.col],
        [AllDiff(grid[x:x+N, y:y+N]) for x in range(0, N*N, N) for y in range(0, N * N, N)],
        [(x == int(v)) for x, v in zip(grid.flat, ''.join(open(clues)).split()) if v != '*']
    )
    return grid, sudoku

def solve(param):
    N = param['N']
    clues = param['file']
    grid, sudoku = get_model(N, clues)
    solver = sudoku.load(param['solver'])
    solver.solve()
```

Declare the variables in an NxN matrix of cells

State the rules/constraints of Sudoku

Press the “solve” button and a solution is found for you “automagically”
CP = Model (+ Search)

Model description:
user API for declarative programming

The algorithmic part:
finding a solution that satisfies all the constraints, etc, usually by exploring a search tree
A *model* of a constraint satisfaction problem has:

- Variables with sets of possible values, called *domains*:
  - Generally integer sets or integer intervals, such as \( x \in \{5,9,10\} \), but also on floats, graphs, etc.

- Constraints on the variables:
  - Arithmetic  \( 3x + 10y = z \) (linear constraints are a special case!)
  - Logical  \( x < y \) or \( x > z \) (predicate logic)
  - Combinatorial  \( \text{Circuit}(x_1,\ldots,x_n) \) (structural requirements)
Variables: Example

- **Variable** = a decision that should be made.
- **Domain** = finite set of possible values for the variable.
- **Example:**
  - $x_i$ = the city to visit after city $i$ in a tour for the traveling salesperson (TSP);
  - $D(x_i) = \{0, 1, \ldots, i - 1, i + 1, \ldots, n - 1\}$, where $n = \#\text{cities}$: all the possible values for $x_i$. 
### Constraints: Examples

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Logical</th>
<th>Combinatorial</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Sum}(x[], y) \equiv \left( \sum_{i} x_i \right) = y$</td>
<td>$y_i = c \iff y_{ic} = 1$</td>
<td>$\text{AllDifferent}(x[])$</td>
</tr>
</tbody>
</table>

- **Arithmetic**
  - $\text{Sum}(x[], y) \equiv \left( \sum_{i} x_i \right) = y$

- **Logical**
  - $y_i = c \iff y_{ic} = 1$

- **Combinatorial**
  - $\text{AllDifferent}(x[])$
Constraint Programming

- Declarative modeling methodology:
  - Describe properties of a feasible solution.
  - Convey the structure of the problem as explicitly as possible.
  - Express substructures of the problem.
  - Give solvers as much information as possible.

- Computational paradigm:
  - Use constraints to reduce the domain of each variable.
  - Remove from domains some / the values that cannot appear in any solution.
Application Domains

Scheduling

Routing

Rostering

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Minimum consecutive free days for Bath: 2
Day off with Bath: Sunday
After a night shift sequence: 2 free days
Unwanted pattern: E-E-E

Time
A Combinatorial Constraint for Jobshop?

Yes!
– Disjunctive(...)
A Combinatorial Constraint for Jobshop?

Yes!
– Disjunctive(…)

\[
D(\text{start}[i]) = \{0, \ldots, H\}
\]
\[
\text{end}[i] = \text{start}[i] + \text{duration}[i]
\]
Why CP?

– Small model.
– Rich modeling language.

– One can model idiosyncratic requirements with “side constraints”.
– CP can be used in hybrid solvers.
– One can program the search and exploit problem knowledge.

– Rich constraint catalog (Global Constraint Catalog has ~400 constraints).
– Default search available, too.
– High-level domains (e.g., sets, graphs, bit vectors, …).
TSP Modeling: CP vs MIP

**MIP**

minimize \( \sum_{i,j} d_{ij} \cdot x_{ij} \)

subject to \( \sum_{i \in V} x_{ij} = 2 \forall i \in V \)

\( \sum_{i,j \in S, i \neq j} x_{ij} \leq |S| - 1 \forall S \subset V, S \neq \phi \)

\( x_{ij} \in \{0,1\} \)

**CP**

minimize \( \sum_{i \in V} d_{i,succ[i]} \)

subject to Circuit(succ)

\( succ[i] \in \{0,\ldots, i - 1, i + 1, n - 1\} \)

![Diagram](https://via.placeholder.com/150)
TSP Modeling: CP vs MIP

**MIP**

Minimize: \( \sum_{i,j} d_{ij} \cdot x_{ij} \)

Subject to:

- \( \sum_{i \in V} x_{ij} = 2 \quad \forall i \in V \)
- \( \sum_{i,j \in S, i \neq j} x_{ij} \leq |S| - 1 \quad \forall S \subset V, S \neq \emptyset \)
- \( x_{ij} \in \{0,1\} \)

**CP**

Minimize: \( \sum_{i \in V} d_{i, succ[i]} \)

Subject to Circuit\((succ)\)

\( succ[i] \in \{0, \ldots, i - 1, i + 1, n - 1\} \)
Constraint Programming

Computation
Constraint Programming

▶ Computational paradigm:
  • Use constraints to reduce the domain of each variable.
  • Make a search choice when no more domain reduction can be performed.

▶ What is a search choice?
  • There are many possible search choices!
  • For the moment, assume a search choice fixes a variable to a value.

▶ Search choices can be wrong:
  • In optimization, search choices are often wrong :-(
  • The solver then backtracks, i.e., it tries another value.
Coloring a Map

Specification:
- Color a map so that no two adjacent territories have the same color.

The 4 Color Theorem:
- Every map can be colored with just 4 colors.
- Proven by Kenneth Appel and Wolfgang Haken.
- First major theorem proven with a computer.
Coloring a Map

▪ How to color a map with constraint programming?
  – Choose the variables.
  – Express the constraints in terms of the variables.

▪ What are the variables?
  – A variable for each territory, denoting its color.

▪ What are the domains of the variables?
  – The set of values that a variable can take: the four colors.

▪ How to express the constraints?
  – State that two adjacent territories cannot be given the same color.
enum Countries = { Belgium, Denmark, France, Germany, Netherlands, Luxembourg }; 
enum Colors = { black, yellow, red, blue }; 
var{Colors} color[Countries]; 

solve { 
    color[Belgium] ≠ color[France]; 
    color[Belgium] ≠ color[Germany]; 
    color[Belgium] ≠ color[Netherlands]; 
    color[Belgium] ≠ color[Luxembourg]; 
    color[Denmark] ≠ color[Germany]; 
    color[France] ≠ color[Germany]; 
    color[France] ≠ color[Luxembourg]; 
    color[Germany] ≠ color[Netherlands]; 
    color[Germany] ≠ color[Luxembourg]; 
}
Coloring a Map

- Denmark
- Netherlands
- Belgium
- Luxembourg
- France
- Germany
Coloring a Map
Coloring a Map
Coloring a Map

```c
enum Countries = { Belgium, Denmark, France, Germany, Netherlands, Luxembourg };
enum Colors = { black, yellow, red, blue }; var{Colors} color[Countries];
solve {
    color[Belgium] ≠ color[France];
    color[Belgium] ≠ color[Germany];
    color[Belgium] ≠ color[Netherlands];
    color[Belgium] ≠ color[Luxembourg];
    color[Denmark] ≠ color[Germany];
    color[France] ≠ color[Germany];
    color[France] ≠ color[Luxembourg];
    color[Germany] ≠ color[Netherlands];
    color[Germany] ≠ color[Luxembourg];
}
```
enum Countries = { Belgium, Denmark, France, Germany, Netherlands, Luxembourg }; 
enum Colors = { black, yellow, red, blue }; 
var{Colors} color[ Countries ]; 
solve { 
black ≠ color[ France ]; 
black ≠ color[ Germany ]; 
black ≠ color[ Netherlands ]; 
black ≠ color[ Luxembourg ]; 
color[ Denmark ] ≠ color[ Germany ]; 
color[ France ] ≠ color[ Germany ]; 
color[ France ] ≠ color[ Luxembourg ]; 
color[ Germany ] ≠ color[ Netherlands ]; 
color[ Germany ] ≠ color[ Luxembourg ]; 
}
Coloring a Map
Coloring a Map
Coloring a Map
Coloring a Map
Coloring a Map
Coloring a Map
Coloring a Map
Coloring a Map
Computational Paradigm

- Branch and prune:
  - Pruning:
    • Reduce the search space as much as possible.
  - Branching = making search choices:
    • Decompose the problem into subproblems and explore the subproblems.
- Pruning (aka filtering, propagating, contracting, narrowing):
  • Use constraints to remove, from the variable domains, values that cannot belong to any solution.
- Branching:
  • For example, try all the domain values of a variable until either a solution is found or it can be proven that no solution exists.
Constraint Programming

Search and Fixpoint Algorithm
Computational Paradigm

‣ Complete method, not a heuristic, because a search-tree exploration:
  – Given enough time, it will find a / all solution(s) to a satisfaction problem.
  – Given enough time, it will find an optimal solution to an optimization problem.

‣ Focus on feasibility:
  – How to use constraints to prune the search space by removing domain values that cannot belong to any solution?
Computational Paradigm

- Search
- Constraint Store

?
Computational Paradigm

Search

Constraint Store

X[0]=0

x[0]=0
Computational Paradigm

Search

Constraint Store

Success

X[0]=0
Computational Paradigm

Search

X[0]=0
X[1]=2
Success

Constraint Store

x[0]=0
x[1]=2

Success
Computational Paradigm

Search

X[0]=0

X[1]=2

Success

Failure

Constraint Store

x[0]=0

x[1]=2
Computational Paradigm

Search

Constraint Store

Constraint

Domain Store

C_1

C_2

C_3

C_4

C_5
Computational Paradigm

▪ What does a constraint do?
  – Feasibility checking
  – Pruning
▪ Feasibility checking:
  – Can the constraint be satisfied given the values in the domains of its variables?
▪ Pruning:
  – If satisfiable = feasible, then a constraint removes values in the domains that cannot be part of any solution.
The propagation engine:
- This is the core of any constraint-programming solver.
- It is a simple fixpoint algorithm:

```plaintext
fixPoint()
{
    repeat
        select a constraint c;
        if c is infeasible given the domain store then
            return failure;
        else
            apply the pruning algorithm associated with c;
        until (no constraint can remove any value from the domain of any of its variables);
    return success;
}
```
Computational Paradigm

Search

add constraints

fixPoint()

Constraint

Store

C_1

C_2

C_3

C_4

C_5

Constraint

Domain

Store
When to Execute the Fixpoint Algorithm?

```plaintext
fixPoint()
{
    repeat
        select a constraint c;
        if c is infeasible given the domain store then
            return failure;
        else
            apply the pruning algorithm associated with c;
        until (no constraint can remove any value ...);
    return success;
}
```
When to Execute the Fixpoint Algorithm?

```
fixPoint()
{
    repeat
    select a constraint c;
    if c is infeasible given the domain store then
    return failure;
    else
    apply the pruning algorithm associated with c;
    until (no constraint can remove any value ...);
    return success;
}
```

Initially, and whenever a constraint is added during the search!
8-Queens: Model

 Representation:
 – An array of integer variables, one per column, denoting the row of its queen.

 Constraints:
 – No two queens can be placed on the same
   - row
   - upward diagonal
   - downward diagonal
8-Queens: Model

- Representation:
  - An array of integer variables, one per column, denoting the row of its queen.

- Constraints:
  - No two queens can be placed on the same row, upward diagonal, or downward diagonal.

```plaintext
range R = 1..8;
var{int} q[R] in R;
solve {
  forall(i in R, j in R: i < j) {
    q[i] ≠ q[j];
    q[i] ≠ q[j] + (j - i);
    q[i] ≠ q[j] - (j - i);
  }
}
```
8-Queens: Model

- **Representation:**
  - An array of integer variables, one per column, denoting the row of its queen.

- **Constraints:**
  - No two queens can be placed on the same row, upward diagonal, or downward diagonal.

```plaintext
range R = 1..8;
var{int} q[R] in R;
solve {
    forall(i in R, j in R: i < j) {
        q[i] ≠ q[j];
        q[i] ≠ q[j] + (j - i);
        q[i] ≠ q[j] - (j - i);
    }
}
```
8-Queens: Model

- **Representation:**
  - An array of integer variables, one per column, denoting the row of its queen.

- **Constraints:**
  - No two queens can be placed on the same row
  - upward diagonal
  - downward diagonal

```plaintext
range R = 1..8;
var{int} q[R] in R;
solve {
    forall(i in R, j in R: i < j) {
        q[i] ≠ q[j];
        q[i] ≠ q[j] + (j - i);
        q[i] ≠ q[j] - (j - i);
    }
}
```
8-Queens: Model

- **Representation:**
  - An array of integer variables, one per column, denoting the row of its queen.

- **Constraints:**
  - No two queens can be placed on the same row
  - upward diagonal
  - downward diagonal

```plaintext
range R = 1..8;
var {int} q[R] in R;
solve {
  forall(i in R, j in R: i < j) {
    q[i] ≠ q[j];
    q[i] ≠ q[j] + (j - i);
    q[i] ≠ q[j] - (j - i);
  }
}
```
8-Queens: Model

- **Representation:**
  - An array of integer variables, one per column, denoting the row of its queen.

- **Constraints:**
  - No two queens can be placed on the same
    - row
    - upward diagonal
    - downward diagonal

```plaintext
range R = 1..8;
var {int} q[R] in R;
solve {
  forall (i in R, j in R : i < j) {
    q[i] ≠ q[j];
    q[i] ≠ q[j] + (j - i);
    q[i] ≠ q[j] - (j - i);
  }
}
```

A solution to 4-Queens
8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);
    }
}
```
8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);
    }
}
```
8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);
    }
}
```
8-Queens: Model in MiniCP

– Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);
    }
}
```
8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);
    }
}
```
8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {

        }
    }
}
```

- Cannot be in the same column...

8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);

        // Cannot be in the same column...
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
            }
    }
}
```
8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
                cp.post(Factory.notEqual(q[i], q[j]));
            }  
    }
```
8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
                cp.post(Factory.notEqual(q[i], q[j]));
            }
    }
}
```

- Cannot be on the same row...
– Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
                cp.post(Factory.notEqual(q[i], q[j]));
                cp.post(Factory.notEqual(q[i], q[j], j - i));
            }
    }
}
```

– Cannot be on the same diagonal…
8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
                cp.post(Factory.notEqual(q[i], q[j]));
                cp.post(Factory.notEqual(q[i], q[j], j - i));
            }
    }
}
```

- Cannot be on the same diagonal...
8-Queens: Model in MiniCP

- Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
                cp.post(Factory.notEqual(q[i], q[j]));
                cp.post(Factory.notEqual(q[i], q[j], j - i));
                cp.post(Factory.notEqual(q[i], q[j], i - j));
            }
    }
}
```

- Cannot be on the same diagonals...
– Holds an array of integer variables with one variable per column.

```java
public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);

        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
                cp.post(Factory.notEqual(q[i], q[j]));
                cp.post(Factory.notEqual(q[i], q[j], j - i));
                cp.post(Factory.notEqual(q[i], q[j], i - j));
            }
    }
}
```

– Cannot be on the same diagonals...
import static minicp.cp.Factory.*;

public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
                cp.post(notEqual(q[i], q[j]));
                cp.post(notEqual(q[i], q[j], j - i));
                cp.post(notEqual(q[i], q[j], i - j));
            }
        DFSSearch dfs = makeDfs(cp, () -> {
            int idx = -1;
            for (int k = 0; k < q.length; k++)
                if (q[k].size() > 1) { idx = k; break; }
            if (idx == -1) return new Procedure[0];
            else {
                IntVar qi = q[idx];
                int v = qi.min();
                Procedure left = () -> cp.post(equal(qi, v));
                Procedure right = () -> cp.post(notEqual(qi, v));
                return new Procedure[] {left, right};
            }
        });
        dfs.solve();
    }
}
import static minicp.cp.Factory.*;

public class NQueens {
    public static void main(String[] args) {
        int n = 8;
        Solver cp = Factory.makeSolver(false);
        IntVar[] q = Factory.makeIntVarArray(cp, n, n);
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++) {
                cp.post(notEqual(q[i], q[j]));
                cp.post(notEqual(q[i], q[j], j - i));
                cp.post(notEqual(q[i], q[j], i - j));
            }
        DFSearch dfs = makeDfs(cp, () -> {
            int idx = -1;
            for (int k = 0; k < q.length; k++)
                if (q[k].size() > 1) { idx = k; break; }
            if (idx == -1) return new Procedure[0];
            else {
                IntVar qi = q[idx];
                int v = qi.min();
                Procedure left = () -> cp.post(equal(qi, v));
                Procedure right = () -> cp.post(notEqual(qi, v));
                return new Procedure[] { left, right };}
        });
        dfs.solve();
    }
}
4-Queens: Finding the First Solution

Current State

Search Tree

4-Queens: Finding the First Solution

- q[0] = 0
- q[0] ≠ 0

fixed variable:
singleton domain
4-Queens: Finding the First Solution

- $q[0] = 0$
- $q[0] \neq 0$

- Fixed variable: singleton domain

- Grid representation:
  - $q[0]$ to $q[3]$
  - Positions:
    - $q[0]$ in row 0
    - $q[1]$ in row 1
    - $q[2]$ in row 2
    - $q[3]$ in row 3

4-Queens: Finding the First Solution

$q[0]=0$
$q[1]=2$
$q[0]!=0$
$q[1]!=2$
4-Queens: Finding the First Solution

q[0] = 0
q[1] = 2
q[0]! = 0
q[1]! = 2


0 1 2 3
4-Queens: Finding the First Solution

q[0] = 0
q[0] != 0
q[1] = 2
q[1] != 2

empty domain: wipe-out!
4-Queens: Finding the First Solution

- $q[0]=0$
- $q[1]=2$
- $q[0]! = 0$
- $q[1]! = 2$

Grid:

```
  0 1 2 3
```
4-Queens: Finding the First Solution

\[ q[0] = 0 \quad q[0] \neq 0 \]
\[ q[1] = 2 \quad q[1] \neq 2 \]


0 1 2 3
4-Queens: Finding the First Solution

- \( q[0] = 0 \)
- \( q[1] = 2 \)
- \( q[1]! = 2 \)
- \( q[0]! = 0 \)

Grid representation:

- Row 0
- Row 1
- Row 2
- Row 3

- Square 0
- Square 1
- Square 2
- Square 3

- Queens at positions (0, 2) and (2, 0)
4-Queens: Finding the First Solution

$q[0]=0$  $q[0]! = 0$

$q[1]=2$  $q[1]! = 2$

4-Queens: Finding the First Solution

q[0] = 0
q[0] ≠ 0
q[1] = 2
q[1] ≠ 2
q[0] ≠ 0
q[1] ≠ 2
4-Queens: Finding the First Solution

- \( q[0] = 0 \)
- \( q[0] \neq 0 \)
- \( q[1] = 2 \)
- \( q[1] \neq 2 \)

- Empty domain!
4-Queens: Finding the First Solution

$q[0] = 0$
$q[0]! = 0$
$q[1] = 2$
$q[1]! = 2$

0
1
2
3
4-Queens: Finding the First Solution

\[ q[0] = 0 \]
\[ q[1] = 2 \]
\[ q[0]! = 0 \]
\[ q[0]! = 1 \]
\[ q[1]! = 2 \]
\[ q[0] = 1 \]
4-Queens: Finding the First Solution

\[ q[0] = 0 \]
\[ q[1] = 2 \]
\[ q[0]! = 0 \]
\[ q[1]! = 2 \]
\[ q[0]! = 1 \]
\[ q[0]! = 1 \]
4-Queens: Finding the First Solution

\[
\begin{align*}
q[0] &= 0 \\
q[1] &= 2 \\
q[0]! &= 1 \\
q[0]! &= 1
\end{align*}
\]
4-Queens: Finding the First Solution

- \( q[0] = 0 \)
- \( q[0] \neq 0 \)
- \( q[1] = 2 \)
- \( q[1] \neq 2 \)
- \( q[0] = 1 \)
- \( q[0] \neq 1 \)

```
```

0
1
2
3
4-Queens: Finding the First Solution

- \( q[0] = 0 \)
- \( q[1] = 2 \)
- \( q[0]! = 0 \)
- \( q[1]! = 2 \)
- \( q[0] = 1 \)
- \( q[0]! = 1 \)
- \( q[2] \)
- \( q[3] \)
4-Queens: Finding the First Solution

Solution
8-Queens: Search

The purpose of the branching scheme:
- Build the search tree.
- Applied at every node:
  - Either it yields nothing: the search is done.
  - Or it yields a set of constraints: there is one sub-problem per constraint.
- Represented as a first-order function (a lambda expression, aka a closure).

Bottomline:
- This is a **BRANCH & PRUNE** framework.

Decompose & Explore  Reason: remove values that cannot appear in solutions
Computational Paradigm

Search

Constraint Store
**Computational Paradigm**

- Search
- Constraint Store

\[ X = 5 \]
Computational Paradigm

Search

$X = 5$

Constraint Store

Success
Computational Paradigm

Search

\[ X = 5 \]

\[ Y \neq 2 \]

Constraint Store

Success
Computational Paradigm

Search

$X = 5$

$Y \neq 2$

Constraint Store

Success

Failure
Computational Paradigm

Search

Constraint Store

Constraint

Domain Store

C1

C2

C3

C4

C5
Constraint Contract

- What does a constraint do?
  - Feasibility checking.
  - Pruning.

- Feasibility checking:
  - Can the constraint be satisfied given the values in the domains of its variables?
  - Example: $X \in \{1,3,5\}, Y \in \{3,5\}$: constraint $X \neq Y$ is feasible.

- Pruning:
  - If satisfiable, remove values in the domains that cannot be part of any solution.
  - Example: $X \in \{1,3,5\}, Y \in \{5\}$: constraint $X \neq Y$ can remove 5 from domain of $X$.

- The framework uses dedicated algorithms for each constraint:
  - A *filtering algorithm* exploits the structure and properties of its constraint.
Constraint Programming

Send More Money
Send More Money

- Basic model:
  - Add carries explicitly, like in primary school.

- What are the variables?
  - There is a variable for each letter, in order to denote the values of the letters.
  - There is a variable for each carry.
Send More Money: Model

enum Letters = { S, E, N, D, M, O, R, Y};
range Digits = 0..9;
var{int} value[Letters] in Digits;
var{int} carry[1..4] in 0..1;

solve {
  forall (i in Letters, j in Letters: i < j)
    value[i] ≠ value[j];
  value[S] ≠ 0;
  value[M] ≠ 0;
  carry[4] = value[M];
  value[D] + value[E] = value[Y] + 10 * carry[1];
}

Domain(value[i]) = Digits, ∀i∈Letters
What is the Search Space?
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

forall(i in Letters, j in Letters: i < j)
value[i] ≠ value[j];
value[S] ≠ 0;
value[M] ≠ 0;
carry[4] = value[M];
What is the Search Space?

forall(i in Letters, j in Letters: i < j)
   value[i] ≠ value[j];
value[S] ≠ 0;
value[M] ≠ 0;
carry[4] = value[M];
What is the Search Space?

forall(i in Letters, j in Letters: i < j)
value[i] \neq value[j];
value[S] \neq 0;
value[M] \neq 0;
carry[4] = value[M];
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

forall(i in Letters, j in Letters: i < j)
value[i] ≠ value[j];
value[S] ≠ 0;
value[M] ≠ 0;
carry[4] = value[M];
What is the Search Space?

forall(i in Letters, j in Letters: i < j) value[i] ≠ value[j];
value[S] ≠ 0;
value[M] ≠ 0;
carry[4] = value[M];
What is the Search Space?

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

forall(i in Letters, j in Letters: i < j)
value[i] ≠ value[j];
value[S] ≠ 0;
value[M] ≠ 0;
carry[4] = value[M];
### What is the Search Space?

#### Constraints:

```plaintext
forall(i in Letters, j in Letters: i < j)
  value[i] ≠ value[j];
value[S] ≠ 0;
value[M] ≠ 0;
carry[4] = value[M];
```
What is the Search Space?

forall(i in Letters, j in Letters: i < j)
   value[i] ≠ value[j];
value[S] ≠ 0;
value[M] ≠ 0;
carry[4] = value[M];
What is the Search Space?

forall(i in Letters, j in Letters: i < j) value[i] != value[j];
value[S] != 0;
value[M] != 0;
carry[4] = value[M];
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

forall(i in Letters, j in Letters: i < j)
value[i] ≠ value[j];
value[S] ≠ 0;
value[M] ≠ 0;

\[\text{carry}[4] = \text{value}[M];\]
What is the Search Space?

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{carry}[3] + \text{value}[S] + \text{value}[M] = \text{value}[O] + 10 \times \text{carry}[4];
\]
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is the Search Space?

\[
\text{carry}[3] + \text{value}[S] + \text{value}[M] = \text{value}[O] + 10 \times \text{carry}[4];
\]

\[
\text{carry}[3] + \text{value}[S] + 1 = \text{value}[O] + 10 \times \text{carry}[4];
\]
What is the Search Space?

\[
\begin{align*}
\text{carry}[3] + \text{value}[S] + \text{value}[M] &= \text{value}[O] + 10 \times \text{carry}[4]; \\
\text{carry}[3] + \text{value}[S] + 1 &= \text{value}[O] + 10 \times \text{carry}[4];
\end{align*}
\]
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( \text{carry}[3] + \text{value}[S] + \text{value}[M] = \text{value}[O] + 10 \times \text{carry}[4]; \)
- \( \text{carry}[3] + \text{value}[S] + 1 = \text{value}[O] + 10 \times \text{carry}[4]; \)

\( \in [3..11] \)

\( \in [0 + 2 + 1..1 + 9 + 1] \)

\( \in [10..19] \)
What is the Search Space?

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{carry}[3] + \text{value}[S] + \text{value}[M] = \text{value}[O] + 10 \times \text{carry}[4];
\]
\[
\text{carry}[3] + \text{value}[S] + 1 = \text{value}[O] + 10 \times \text{carry}[4];
\]

\[\in [3..11]\]
\[\in [0 + 2 + 1..1 + 9 + 1]\]
\[\in [10..19]\]
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{l}
\text{carry}[3] + \text{value}[S] + \text{value}[M] = \text{value}[O] + 10 \times \text{carry}[4];
\\
\text{carry}[3] + \text{value}[S] + 1 = \text{value}[O] + 10 \times \text{carry}[4];
\end{array}
\]

\[\in [3..11]\]
\[\in [0 + 2 + 1..1 + 9 + 1]\]
\[\in [10..19]\]
What is the Search Space?

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{carry}[3] + \text{value}[S] + \text{value}[M] &= \text{value}[O] + 10 \times \text{carry}[4]; \\
\text{carry}[3] + \text{value}[S] + 1 &= \text{value}[O] + 10 \times \text{carry}[4]; \\
\end{align*}
\]

\[
\begin{align*}
\in [3..11] \\
\in [0 + 2 + 1..1 + 9 + 1] \\
\in [10..19]
\end{align*}
\]
What is the Search Space?

\[
\begin{align*}
\text{carry}[3] + \text{value}[S] + \text{value}[M] &= \text{value}[O] + 10 \times \text{carry}[4]; \\
\text{carry}[3] + \text{value}[S] + 1 &= \text{value}[O] + 10 \times \text{carry}[4];
\end{align*}
\]

\[
\begin{align*}
\epsilon &\in [3..11] \\
\epsilon &\in [0 + 2 + 1..1 + 9 + 1] \\
\epsilon &\in [10..19]
\end{align*}
\]
What is the Search Space?

\[
\begin{align*}
\text{carry}[3] + \text{value}[S] + \text{value}[M] &= \text{value}[O] + 10 \times \text{carry}[4]; \\
\text{carry}[3] + \text{value}[S] + 1 &= \text{value}[O] + 10 \times \text{carry}[4];
\end{align*}
\]

\[
\begin{align*}
\in [3..11] & \quad \in [0 + 2 + 1..1 + 9 + 1] & \quad \in [10..19]
\end{align*}
\]
What is the Search Space?

\[
\]
\[
\]
\[\in [3..11]\]
\[\in [0 + 2 + 1..1 + 9 + 1]\]
\[\in [10..19]\]
What is the Search Space?

\[
\begin{align*}
\text{carry}[3] + \text{value}[S] + \text{value}[M] &= \text{value}[O] + 10 \times \text{carry}[4]; \\
\text{carry}[3] + \text{value}[S] + 1 &= \text{value}[O] + 10 \times \text{carry}[4];
\end{align*}
\]

\[\in [3..11] \cap [10..19] = [10..11]\]
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{carry}[3] + \text{value}[S] + \text{value}[M] &= \text{value}[O] + 10 \times \text{carry}[4]; \\
\text{carry}[3] + \text{value}[S] + 1 &= \text{value}[O] + 10 \times \text{carry}[4];
\end{align*}
\]

\[
\begin{align*}
\in [3..11] \\

\in [0 + 2 + 1..1 + 9 + 1] \\

\in [10..19]
\end{align*}
\]

\[
[3..11] \cap [10..19] = [10..11]
\]
What is the Search Space?

\[
\begin{align*}
10 \leq carry[3] + value[S] + 1 &\leq 11 \\
\end{align*}
\]

\[ [3..11] \cap [10..19] = [10..11]\]
What is the Search Space?


\[ carry[3] + value[S] + 1 = value[O] + 10 \times carry[4]; \]

\[ 10 \leq carry[3] + value[S] + 1 \leq 11 \]
What is the Search Space?

10 \leq \text{carry}[3] + \text{value}[S] + 1 \leq 11
What is the Search Space?

10 \leq \text{carry}[3] + \text{value}[S] + 1 \leq 11

10 - 1 \leq \text{carry}[3] + \text{value}[S] \leq 11 - 1
What is the Search Space?

\[
10 \leq \text{carry}[3] + \text{value}[S] + 1 \leq 11
\]

\[
10 - 1 \leq \text{carry}[3] + \text{value}[S] \leq 11 - 1
\]

\[
10 - 1 - \text{carry}[3] \leq \text{value}[S] \leq 11 - 1 - \text{carry}[3]
\]
What is the Search Space?

\[
\begin{align*}
10 &\leq carry[3] + value[S] + 1 \leq 11 \\
10 - 1 &\leq carry[3] + value[S] \leq 11 - 1 \\
\end{align*}
\]
What is the Search Space?

\[ 10 \leq \text{carry}[3] + \text{value}[S] + 1 \leq 11 \]

\[ 10 - 1 \leq \text{carry}[3] + \text{value}[S] \leq 11 - 1 \]

\[ 10 - 1 - \text{carry}[3] \leq \text{value}[S] \leq 11 - 1 - \text{carry}[3] \]
What is the Search Space?


10 - 1 <= carry[3] + value[S] <= 11 - 1


10 - 1 - 1 <= value[S] <= 11 - 1 - 0
What is the Search Space?

- $10 \leq \text{carry}[3] + \text{value}[S] + 1 \leq 11$
- $10 - 1 \leq \text{carry}[3] + \text{value}[S] \leq 11 - 1$
- $10 - 1 - \text{carry}[3] \leq \text{value}[S] \leq 11 - 1 - \text{carry}[3]$
- $10 - 1 - 1 \leq \text{value}[S] \leq 11 - 1 - 0$

$8 \leq \text{value}[S] \leq 10$
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \(10 \leq \text{carry}[3] + \text{value}[S] + 1 \leq 11\)
- \(10 - 1 \leq \text{carry}[3] + \text{value}[S] \leq 11 - 1\)
- \(10 - 1 - \text{carry}[3] \leq \text{value}[S] \leq 11 - 1 - \text{carry}[3]\)
- \(10 - 1 - 1 \leq \text{value}[S] \leq 11 - 1 - 0\)
- \(8 \leq \text{value}[S] \leq 10\)
What is the Search Space?

S  E  N  D  M  O  R  Y
C_4 C_3 C_2 C_1

0 1 2 3 4 5 6 7 8 9
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

carry[3] + value[S] + 1 = value[0] + 10 * carry[4];

∈ [10..11]
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{carry}[3] + \text{value}[S] + 1 = \text{value}[O] + 10 \times \text{carry}[4];
\]

\[
10 \leq \text{value}[O] + 10 \times \text{carry}[4] \leq 11
\]
What is the Search Space?

\[
\begin{aligned}
\text{carry}[3] + \text{value}[S] + 1 &= \text{value}[O] + 10 \times \text{carry}[4]; \\
10 \leq \text{value}[O] + 10 \times \text{carry}[4] &\leq 11
\end{aligned}
\]
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{carry}[3] + \text{value}[S] + 1 = \text{value}[O] + 10 \times \text{carry}[4];
\]

\[
10 \leq \text{value}[O] + 10 \times \text{carry}[4] \leq 11
\]

\[
10 \leq \text{value}[O] + 10 \times 1 \leq 11
\]
What is the Search Space?


10 <= value[O] + 10 * 1 <= 11

0 <= value[O] <= 1
What is the Search Space?

\[
\begin{align*}
10 & \leq \text{value}[O] + 10 \times \text{carry}[4] & \leq 11 \\
10 & \leq \text{value}[O] + 10 \times 1 & \leq 11 \\
0 & \leq \text{value}[O] & \leq 1 \\
\end{align*}
\]

The Search Space is defined by the constraints:

- \(10 \leq \text{value}[O] + 10 \times \text{carry}[4] \leq 11\)
- \(10 \leq \text{value}[O] + 10 \times 1 \leq 11\)
- \(0 \leq \text{value}[O] \leq 1\)
What is the Search Space?

\[\begin{align*}
10 &\leq value[O] + 10 \times carry[4] \leq 11 \\
10 &\leq value[O] + 10 \times 1 \leq 11 \\
0 &\leq value[O] \leq 1
\end{align*}\]
What is the Search Space?

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>N</th>
<th>D</th>
<th>M</th>
<th>O</th>
<th>R</th>
<th>C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

```plaintext
63
```

```
10 <= value[O] + 10 * 1 <= 11
0 <= value[O] <= 1
forall(i in Letters, j in Letters)
value[i] != value[j];
```
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
forall(i in Letters, j in Letters) 
value[i] ≠ value[j];
```

```plaintext
```

```plaintext
```

```plaintext
0 <= value[O] <= 1
```

```plaintext
forall(i in Letters, j in Letters) 
value[i] ≠ value[j];
```
Send More Money

```plaintext
enum Letters = { S, E, N, D, M, O, R, Y};
range Digits = 0..9;
var{int} value[Letters] in Digits;
var{int} carry[1..4] in 0..1;

solve {
    forall(i in Letters, j in Letters: i < j)
        value[i] ≠ value[j];
    value[S] ≠ 0;
    value[M] ≠ 0;
    carry[4] = value[M];
    value[D] + value[E] = value[Y] + 10 * carry[1];
}
```

```
C_4  C_3  C_2  C_1
S E N D
+ M O R E
= M O N E Y
```
What is the Search Space?

```
```
What is the Search Space?

What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\]
\[
carry[2] + value[E] + 0 = value[N] + 10 \times carry[3];
\]
What is the Search Space?

\[
\begin{align*}
\text{carry}[2] + \text{value}[E] + \text{value}[O] &= \text{value}[N] + 10 \times \text{carry}[3]; \\
\text{carry}[2] + \text{value}[E] + 0 &= \text{value}[N] + 10 \times \text{carry}[3]; \\
2 \leq \text{value}[N] + 10 \times \text{carry}[3] &\leq 10
\end{align*}
\]
What is the Search Space?

2 \leq \text{value}[N] + 10 \times \text{carry}[3] \leq 10

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 \leq value[N] + 10 \times carry[3] \leq 10

2 - value[N] \leq 10 \times carry[3] \leq 10 - value[N]
What is the Search Space?

\[
2 \leq value[N] + 10 \times carry[3] \leq 10
\]

\[
2 - value[N] \leq 10 \times carry[3] \leq 10 - value[N]
\]
What is the Search Space?

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
2 \leq value[N] + 10 \times carry[3] \leq 10
\]

\[
2 - value[N] \leq 10 \times carry[3] \leq 10 - value[N]
\]
What is the Search Space?

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $2 \leq \text{value}[N] + 10 \times \text{carry}[3] \leq 10$
- $2 - \text{value}[N] \leq 10 \times \text{carry}[3] \leq 10 - \text{value}[N]$
- $-7 \leq 10 \times \text{carry}[3] \leq 8$
### What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $2 \leq \text{value}[N] + 10 \times \text{carry}[3] \leq 10$
- $2 - \text{value}[N] \leq 10 \times \text{carry}[3] \leq 10 - \text{value}[N]$
- $-7 \leq 10 \times \text{carry}[3] \leq 8$
What is the Search Space?

2 \leq \text{value}[N] + 10 \times \text{carry}[3] \leq 10

2 - \text{value}[N] \leq 10 \times \text{carry}[3] \leq 10 - \text{value}[N]

-7 \leq 10 \times \text{carry}[3] \leq 8
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
2 \leq \text{value}[N] + 10 \times \text{carry}[3] \leq 10
\]

\[
2 - \text{value}[N] \leq 10 \times \text{carry}[3] \leq 10 - \text{value}[N]
\]

\[
-7 \leq 10 \times \text{carry}[3] \leq 8
\]
What is the Search Space?

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 <= value[N] + 10 * carry[3] <= 10


-7 <= 10 * carry[3] <= 8
What is the Search Space?

The search space is represented by a grid with the following coordinates:

- **S**: Starting point
- **E**: End point
- **N**: North
- **D**: Down
- **M**: Middle
- **O**: Original
- **R**: Right
- **Y**: Y-axis
- **C4**: Column 4
- **C3**: Column 3
- **C2**: Column 2
- **C1**: Column 1

The grid shows the possible paths and positions within the search space.
What is the Search Space?

def enum Letters = { S, E, N, D, M, O, R, Y};
def range Digits = 0..9;
def var{int} value[Letters] in Digits;
def var{int} carry[1..4] in 0..1;

solve {
    forall(i in Letters, j in Letters: i < j)
        value[i] ≠ value[j];
    value[S] ≠ 0;
    value[M] ≠ 0;
    carry[4] = value[M];
    value[D] + value[E] = value[Y] + 10 * carry[1];
}
enum Letters = { S, E, N, D, M, O, R, Y};
range Digits = 0..9;
var{int} value[Letters] in Digits;
var{int} carry[1..4] in 0..1;
solve {
    forall(i in Letters, j in Letters: i < j)
        value[i] ≠ value[j];
    value[S] ≠ 0;
    value[M] ≠ 0;
    carry[4] = value[M];
    value[D] + value[E] = value[Y] + 10 * carry[1];
}
What is the Search Space?

```
enum Letters = { S, E, N, D, M, O, R, Y};
range Digits = 0..9;
var{int} value[Letters] in Digits;
var{int} carry[1..4] in 0..1;
solve {
    forall(i in Letters, j in Letters: i < j)
        value[i] ≠ value[j];
    value[S] ≠ 0;
    value[M] ≠ 0;
    carry[4] = value[M];
    value[D] + value[E] = value[Y] + 10 * carry[1];
}
```
What is the Search Space?

```
enum Letters = { S, E, N, D, M, O, R, Y};
range Digits = 0..9;
var{int} value[Letters] in Digits;
var{int} carry[1..4] in 0..1;

solve {
    forall(i in Letters, j in Letters: i < j)
    value[i] ≠ value[j];
    value[S] ≠ 0;
    value[M] ≠ 0;
    carry[4] = value[M];
    {carry[3];}:
    value[S] + value[M] = value[O] + 10 * carry[4];
    value[D] + value[E] = value[Y] + 10 * carry[1];
}
```
What is the Search Space?

What is the Search Space?

What is the Search Space?

```

0 + value[S] + 1 = 0 + 10 * 1;
```
What is the Search Space?

\[
\begin{align*}
\text{carry}[3] + \text{value}[S] + \text{value}[M] &= \text{value}[O] + 10 \times \text{carry}[4]; \\
0 + \text{value}[S] + 1 &= 0 + 10 \times 1; \\
\text{value}[S] &= 9;
\end{align*}
\]
What is the Search Space?

\[
\begin{align*}
0 + value[S] + 1 &= 0 + 10 \times 1; \\
value[S] &= 9;
\end{align*}
\]
What is the Search Space?

\[
\text{carry}[3] + \text{value}[S] + \text{value}[M] = \text{value}[O] + 10 \times \text{carry}[4];
\]

\[
0 + \text{value}[S] + 1 = 0 + 10 \times 1;
\]

\[
\text{value}[S] = 9;
\]

\[
\text{forall}(i \in \text{Letters}, j \in \text{Letters}: i < j) \quad \text{value}[i] \neq \text{value}[j];
\]
What is the Search Space?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\]

\[
0 + value[S] + 1 = 0 + 10 \times 1;
\]

\[
value[S] = 9;
\]

\[
\text{forall}(i \text{ in Letters}, j \text{ in Letters}: i < j) \text{ value}[i] \neq \text{value}[j];
\]
What is the Search Space?

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
public class SMoney {
    enum Letters {
        S(0), E(1), N(2), D(3), M(4), O(5), R(6), Y(7);
        public final int val;
        Letters(int v) { val = v; }
    }
    public static void main(String[] args) {
        Solver cp = Factory.makeSolver(false);
        IntVar[] values = Factory.makeIntVarArray(cp, Y.val + 1, 0, 9);
        IntVar[] carry = Factory.makeIntVarArray(cp, 4, 0, 1);

        cp.post(allDifferent(values));
        cp.post(notEqual(values[S.val], 0));
        cp.post(notEqual(values[M.val], 0));
        cp.post(equal(values[M.val], carry[3]));
        cp.post(equal(sum(carry[2], values[S.val], values[M.val], minus(values[O.val]), mul(carry[3], -10)), 0));
        cp.post(equal(sum(carry[1], values[E.val], values[O.val], minus(values[N.val]), mul(carry[2], -10)), 0));
        cp.post(equal(sum(carry[0], values[N.val], values[R.val], minus(values[E.val]), mul(carry[1], -10)), 0));
        cp.post(equal(sum(values[D.val], values[E.val], minus(values[Y.val]), mul(carry[0], -10)), 0));
        search.onSolution(() ->
            System.out.println("solution:" + Arrays.toString(values))
        );
        DFSSearch search = Factory.makeDfs(cp, firstFail(values));
        SearchStatistics stats = search.solve();
        System.out.format("#Solutions: %s\n", stats.numberOfSolutions());
        System.out.format("Statistics: %s\n", stats);
    }
}
Constraint Programming

Constraint Satisfaction Problem (CSP)
Constraint Satisfaction Problem (CSP)

- CSP = a triplet <X,D,C>
- X = set of variables, each variable x with its domain D(x)
- D = Cartesian product of all the domains
- C = set of constraints

Example:
- c1 = AllDifferent(x1,x2,x3,x4)
- Vars(c1) = \{x1,x2,x3,x4\}
- Vars(c1) is also called the *scope* of c1
Solution: Definitions

– A candidate solution $\sigma$ associates to each variable a value in its domain:

$$\forall x \in X : \sigma(x) \in D(x)$$

– Evaluation (true/false) of whether a constraint $c$ holds with respect to a candidate solution $\sigma$:

$$c(\sigma) = c(\sigma(x_i^0), \ldots, \sigma(x_{i^{k-1}}))$$

– A solution is a candidate solution that satisfies the evaluations of all constraints in the CSP:

$$\land_{c \in C} c(\sigma) = \text{true}$$

– The set of all solutions to a CSP:

$$\mathcal{S}(\langle X, D, C \rangle)$$
A *filtering algorithm* of a constraint $c$, denoted $\mathcal{F}_c$, is a function on the domains that...

$$\forall D : \mathcal{F}_c(D) \subseteq D \land \mathcal{S}(\langle Vars(c), D, \{c\} \rangle) = \mathcal{S}(\langle Vars(c), \mathcal{F}_c(D), \{c\} \rangle)$$

...shrinks the domains of its variables

– in order to remove values that cannot appear in any solution

– without losing any solution
Data: The CSP \( \langle X, D^0, C \rangle \)
Result: The greatest fixpoint domain

\[
\text{pruningNeeded} \leftarrow \text{true} \\
D \leftarrow D^0 \\
\text{while} \ \text{pruningNeeded} \ \text{do} \\
\quad D^p \leftarrow F_C(D) \\
\quad \text{pruningNeeded} \leftarrow D^p \neq D \\
\quad D \leftarrow D^p \\
\text{end}
\]
Fixpoint Algorithm

```
fixPoint()
{
    repeat
        select a constraint c;
        if c is infeasible given the domain store then
            return failure;
        else
            apply the pruning algorithm associated with c;
        until (no constraint can remove any value);
    return success;
}
```

Data: The CSP \( \langle X, D^0, C \rangle \)
Result: The greatest fixpoint domain

\[
\text{pruningNeeded} \leftarrow \text{true}
\]
\[
D \leftarrow D^0
\]

while \( \text{pruningNeeded} \) do
\[
D^p \leftarrow F_C(D)
\]
\[
\text{pruningNeeded} \leftarrow D^p \neq D
\]
\[
D \leftarrow D^p
\]
end

If no domain of a variable of the constraint c was changed since last time it was executed, is it worth executing it again?
Improved Fixpoint Algorithm: Data-Driven

- The first algorithm is “naïve”:
  - It invokes $\mathcal{F}_c$ on every constraint $c$ all the time.
- We can make this far better!

```
Data: a CSP $\langle X, D^0, C \rangle$
Result: the greatest fixpoint of the filtering algorithms for the constraints in $C$, starting from the domains $D^0$ of the variables of $X$

$Q \leftarrow C$
$D \leftarrow D^0$

while $|Q| > 0$ do
  $c \leftarrow$ dequeue($Q$)
  $D' \leftarrow \mathcal{F}_c(D)$
  $V \leftarrow \{x \in Vars(c) : D'(x) \neq D(x)\}$
  if $|V| > 0$ then
    enqueue($Q$, $\{c' \in C : |Vars(c') \cap V| > 0\}$)
  $D \leftarrow D'$
```
The first algorithm is “naïve”:
- It invokes $\mathcal{F}_c$ on every constraint $c$ all the time.

We can make this far better!

Data: a CSP $\langle X, D^0, C \rangle$
Result: the greatest fixpoint of the filtering algorithms for the constraints in $C$, starting from the domains $D^0$ of the variables of $X$

\[
\begin{align*}
Q & \leftarrow C \\
D & \leftarrow D^0 \\
\text{while } |Q| > 0 & \text{ do} \\
& c \leftarrow \text{dequeue}(Q) \\
& D' \leftarrow \mathcal{F}_c(D) \\
& V \leftarrow \{x \in \text{Vars}(c) : D'(x) \neq D(x)\} \\
& \text{if } |V| > 0 \text{ then} \\
& \quad \text{enqueue}(Q, \{c' \in C : |\text{Vars}(c') \cap V| > 0\}) \\
& \quad D \leftarrow D'
\end{align*}
\]
public class MiniCP implements Solver {
    // ... incomplete ...
    private Queue<Constraint> propagationQueue = new ArrayDeque<>();
    @Override
    public void fixPoint() {
        try {
            notifyFixPoint();
            while (!propagationQueue.isEmpty()) {
                propagate(propagationQueue.remove());
            }
        } catch (InconsistencyException e) {
            // empty the queue and unset the scheduled status
            while (!propagationQueue.isEmpty())
                propagationQueue.remove().setScheduled(false);
            throw e;
        }
    }
}
public class MiniCP implements Solver {
    // ... incomplete ...
    private Queue<Constraint> propagationQueue = new ArrayDeque<>();
    @Override
    public void fixPoint() {
        try {
            notifyFixPoint();
            while (!propagationQueue.isEmpty()) {
                propagate(propagationQueue.remove());
            }
        } catch (InconsistencyException e) {
            // empty the queue and unset the scheduled status
            while (!propagationQueue.isEmpty()) {
                propagationQueue.remove().setScheduled(false);
            }
            throw e;
        }
    }
}

Side effect: enqueue the constraints in propagationQueue that have some variable with a domain change. Details later…
The filtering algorithms are critical:

- For a constraint $c$, the filtering algorithm $\mathcal{F}_c(D)$ is responsible for shrinking $D$.
- CP solvers have a filtering algorithm for each constraint!
- $\mathcal{F}_c(D)$ is specialized to $c$. 
The filtering algorithms are critical:
- For a constraint $c$, the filtering algorithm $\mathcal{F}_c(D)$ is responsible for shrinking $D$.
- CP solvers have a filtering algorithm for each constraint!
  - $\mathcal{F}_c(D)$ is specialized to $c$.

The filtering power can be chosen:
- For a given constraint $c$, there could be several filtering algorithms: $\{\mathcal{F}_c^1, \mathcal{F}_c^2, \ldots\}$
- A filtering power is a compromise between:
  - How much is filtered [contraction]
  - How long it takes to filter [runtime]
Constraint Programming

Filtering Powers
Constraint $X = Y + 1$

Set of solutions for $D(X) = \{1..9\}$ and $D(Y) = \{0..8\}$:
Filtering $X = Y + 1$: Reasoning on the Bounds

$\mathcal{F}_c(D)(x) = \{ v \in D(x) \mid \min(D(y)) + 1 \leq v \leq \max(D(y)) + 1 \}$

$\mathcal{F}_c(D)(y) = \{ v \in D(y) \mid \min(D(x)) - 1 \leq v \leq \max(D(x)) - 1 \}$

Remark: You here need to do it 2x to reach the fixpoint of this rule!
Filtering $X = Y + 1$: Reasoning on the Domains

\[ \mathcal{F}_c(D)(x) = \{ v \in D(x) \mid v - 1 \in D(y) \} \]

\[ \mathcal{F}_c(D)(y) = \{ v \in D(y) \mid v + 1 \in D(x) \} \]

Remove all the points that do not participate in a solution to the constraint, given the domains. Naïve way:
Step 1: identify all the solutions
Step 2: project onto the domains.
Filtering Powers

- Bound consistency: reason with the domain bounds only.
- Domain consistency: reason with all the domain values.

Which one to prefer? It depends on the application!

- Bound consistency:
  - Faster per execution: complexity depends on the number of variables.
  - Prunes less: larger search tree.

- Domain consistency:
  - Slower per execution: complexity depends on the sizes of the domains.
  - Prunes more: smaller search tree.
public class EqualPlusOne extends AbstractConstraint {
    private final IntVar x, y; // x = y + 1

    @Override
    public void post() {
        if (y.isFixed())
            x.fix(y.min() + 1);
        else if (x.isFixed())
            y.fix(x.min() - 1);
        else {
            boundsIntersect();
            x.whenBoundChange(() -> {
                boundsIntersect();
            });
            y.whenBoundChange(() -> {
                boundsIntersect();
            });
        }
    }

    private void boundsIntersect() {
        int newMinX = Math.max(x.min(), y.min() + 1);
        int newMaxX = Math.min(x.max(), y.max() + 1);
        int newMinY = Math.max(y.min(), x.min() - 1);
        int newMaxY = Math.min(y.max(), x.max() - 1);
        x.removeBelow(newMinX);
        x.removeAbove(newMaxX);
        y.removeBelow(newMinY);
        y.removeAbove(newMaxY);
    }
}
public class EqualPlusOne extends AbstractConstraint {
    private final IntVar x, y; // x = y + 1

    @Override
    public void post() {
        if (y.isFixed())
            x.fix(y.min() + 1);
        else if (x.isFixed())
            y.fix(x.min() - 1);
        else {
            boundsIntersect();
            x.whenBoundChange(() -> {
                boundsIntersect();
            });
            y.whenBoundChange(() -> {
                boundsIntersect();
            });
        }
    }

    private void boundsIntersect() {
        int newMinX = Math.max(x.min(), y.min() + 1);
        int newMaxX = Math.min(x.max(), y.max() + 1);
        int newMinY = Math.max(y.min(), x.min() - 1);
        int newMaxY = Math.min(y.max(), x.max() - 1);
        x.removeBelow(newMinX); x.removeAbove(newMaxX);
        y.removeBelow(newMinY); y.removeAbove(newMaxY);
    }
}
X = Y+1 (Bound Consistency)

```java
public class EqualPlusOne extends AbstractConstraint {
    private final IntVar x, y; // x = y + 1

    @Override
    public void post() {
        if (y.isFixed())
            x.fix(y.min() + 1);
        else if (x.isFixed())
            y.fix(x.min() - 1);
        else {
            boundsIntersect();
            x.whenBoundChange(() -> {
                boundsIntersect();
            });
            y.whenBoundChange(() -> {
                boundsIntersect();
            });
        }
    }

    private void boundsIntersect() {
        int newMinX = Math.max(x.min(), y.min() + 1);
        int newX = Math.min(x.max(), y.max() + 1);
        int newMinY = Math.max(y.min(), x.min() - 1);
        int newY = Math.min(y.max(), x.max() - 1);
        x.removeBelow(newMinX); x.removeAbove(newMaxX);
        y.removeBelow(newMinY); y.removeAbove(newMaxY);
    }
}
```

\( \lambda \) called when a bound of \( D(x) \) changes
public class EqualPlusOne extends AbstractConstraint {
    private final IntVar x, y; // x = y + 1

    @Override
    public void post() {
        if (y.isFixed())
            x.fix(y.min() + 1);
        else if (x.isFixed())
            y.fix(x.min() - 1);
        else {
            boundsIntersect();
            x.whenBoundChange(() -> {
                boundsIntersect();
            });
            y.whenBoundChange(() -> {
                boundsIntersect();
            });
        }
    }

    private void boundsIntersect() {
        int newMinX = Math.max(x.min(), y.min() + 1);
        int newXmax = Math.min(x.max(), y.max() + 1);
        int newMinY = Math.max(y.min(), x.min() - 1);
        int newYmax = Math.min(y.max(), x.max() - 1);
        x.removeBelow(newMinX); x.removeAbove(newMaxX);
        y.removeBelow(newMinY); y.removeAbove(newMaxY);
    }
}
So...

What does that mean?
Filtering Power: Discussion
Filtering Power: Discussion

- Strong Filtering: But Slower
- Weak Filtering: But Faster

Filtering power
Filtering Power: Discussion

Strong Filtering
But Slower

Weak Filtering
But Faster

Filtering power

Size of search tree
Filtering Power: Discussion

Strong Filtering But Slower

Weak Filtering But Faster

Filtering power

Size of search tree

Time to explore the tree?
Filtering Power: Discussion

- **Strong Filtering But Slower**
- **Weak Filtering But Faster**

**Filtering power**

**Size of search tree**

**Time to explore the tree?**

**Hard to Predict!**
Recap: Bound and Domain Consistency

Bound Consistency

D(Y)

D(X)

Domain Consistency

D(Y)

D(X)
Bound Consistency: Formal Definition

- Constraint \( c(x_1, \ldots, x_n) \) is *bound consistent* iff \( \forall 1 \leq i \leq n: \)

\[
\exists (v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n), (e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_n) \in [\min(D_1) .. \max(D_1)] \times \ldots \times [\min(D_{i-1}) .. \max(D_{i-1})] \times \ldots \times [\min(D_{i+1}) .. \max(D_{i+1})] \times \ldots \times [\min(D_n) .. \max(D_n)] \text{ such that}
\]

\[
c(v_1, \ldots, \min(D_i), \ldots, v_n) \wedge c(e_1, \ldots, \max(D_i), \ldots, e_n)
\]
Constraint Programming

Constrained Optimization Problem (COP)
Optimization: Definitions

- **Function valuation**
  - of function $f$ with respect to a solution $\sigma$:
    \[
    f(\sigma) = f(\sigma(x_{i_0}), \ldots, \sigma(x_{i_{k-1}}))
    \]

- **Optimal solution**:
  - Yields a valuation better than any other solution:
    \[
    f(\sigma^*) = \min_{\sigma \in \mathcal{S}(\langle X, D, C \rangle)} f(\sigma)
    \]
Branch and Bound with CP

- CP can be used to optimize the value of one variable using a \textit{Branch and Bound} adaptation for CP.

- The DFS looks for all solutions in the search tree, but each time a solution is found (i.e., when every decision variable is fixed to some value) a constraint is dynamically added for the rest of the search tree such that the next-found solution is strictly better.
Example: Minimization Problem

Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.
Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.
Example: Minimization Problem

Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.
Example: Minimization Problem

Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.

Diagram shows a tree structure with nodes numbered from 0 to 553. The objective values decrease as we move down the tree, indicating a minimization problem. The graph on the right plots the objective values against node numbers, showing a clear decrease in values as we explore deeper into the tree.
Example: Minimization Problem

Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.
Example: Minimization Problem

Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.
Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.
Example: Minimization Problem

Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.
Example: Minimization Problem

Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.
Example: Minimization Problem

Each time a solution is found, the next one is strictly better. In the example below, 8 solutions were found before the last, best one. Notice that after a best solution was found, we still explore in order to prove the optimality of this last solution.
Optimization: Implementation

– Assume variable $X$ is minimized:
  the special constraint $X < \textit{bound}$ is added to the constraint store.

– Initially: $\textit{bound} \leftarrow +\infty$.
  If the first-found solution is $X=8$, then the bound is updated, $\textit{bound} \leftarrow 8$, triggering a stronger filtering by that constraint on each node that is still to be explored.

– By construction, the last-found solution is proven to be optimal once the search has been completed.
Branching: 2-step Decision

1. Variable selection
2. Value & partition selection

The branching decisions can have a strong impact on the size of the search tree!
Heuristic Principles
First-fail for variable selection:

Since all variables must eventually be fixed, if there are no solutions under a node (failure), then we prefer to detect this as soon as possible, so that not too much time is spent exploring the subtree under the node.
**Heuristic Principles**

**First-fail** for variable selection:
Since all variables must eventually be fixed, if there are no solutions under a node (failure), then we prefer to detect this as soon as possible, so that not too much time is spent exploring the subtree under the node.

**First-success** for value & partition selection:
Once a variable is selected, say $x_i$, if there is a solution under a node (success), then we want to find it as soon as possible. Since the search tree is explored DFS from left to right, select the **most promising value** $v$ into the domain of $x_i$ on the left branch.
Generic Heuristics

Generic first-fail heuristics (applicable to all problems):

• Select a variable with the smallest domain.
• Select a variable with the smallest difference between its domain bounds.
• Select a most constrained variable (in the scope of the largest number of constraints).
Generic Heuristics

Generic first-fail heuristics (applicable to all problems):

- Select a variable with the smallest domain.
- Select a variable with the smallest difference between its domain bounds.
- Select a most constrained variable (in the scope of the largest number of constraints).

There is no good generic heuristic for the value selection, so we generally try:

- Minimum value of the domain.
- Median value of the domain.
- Maximum value of the domain.
- Problem-dependent heuristic (often a good idea when you can).
Example: Magic Square

Two viewpoints:
· $x_i$ is the number taken by position $i$;
· $y_j$ is the position of the number $j$ (dual variables).

Each value $v$ in $D(x_i)$ corresponds to a variable $y_v$.

Choose a variable $x_i$ with smallest domain (first-fail).
Choose the value $v = \arg\min_{v \in D(x_i)} |D(y_v)|$

Is this first-success?
First-Fail Min-Dom Heuristic for N-Queens

select unfixed variable with smallest domain size
First-Fail Min-Dom Heuristic for N-Queens

```cpp
int n = 8;
Solver cp = makeSolver();
```
First-Fail Min-Dom Heuristic for N-Queens

```cpp
int n = 8;
Solver cp = makeSolver();
IntVar[] q = makeIntVarArray(cp, n, n);
```
int n = 8;
Solver cp = makeSolver();
IntVar[] q = makeIntVarArray(cp, n, n);
for(int i=0;i < n;i++)
    for(int j=i+1;j < n;j++) {
        cp.post(notEqual(q[i],q[j]));
        cp.post(notEqual(q[i],q[j],j-i));
        cp.post(notEqual(q[i],q[j],i-j));
    }
SearchStatistics stats = makeDfs(cp,
    selectMin(q,
        qi -> qi.getSize() > 1,
        qi -> qi.getSize(),
        qi -> {
            int v = qi.getMin();
            return branch(() -> equal(qi,v),
                () -> notEqual(qi,v));
        }
    ).onSolution(() ->
        System.out.println("solution:"+ Arrays.toString(q))
    ).start();

select unfixed variable with smallest domain size
Optimization: Importance of Leftmost Solution

Imagine how small this B&B DFS tree would be if the leftmost solution had objective value 413:
Design a Heuristic to Find Good Leftmost Solutions?

Imagine a greedy algorithm for your problem:

![Diagram showing a decision tree with a greedy decision arrow.

- Leftmost solutions are highlighted in green.
- Rightmost solutions are highlighted in red.
- The arrow indicates the greedy decision point.]
Constraint Programming

MiniCP
This course will use the MiniCP solver:

- Small
- Beautiful
- Efficient
This course will use the MiniCP solver:

- Small
- Beautiful
- Efficient
This course will use the MiniCP solver:

- Small
- Beautiful
- Efficient

- Authors: P. Schaus, P. Van Hentenryck, L. Michel
- Website: http://minicp.org
- Open source: https://github.com/minicp/minicp
- Language: Pure Java (JDK 1.8)
Code base of approx 1,500 lines of Java code:

- Efficiency
- Flexibility
- Simplicity

Solvers try to balance 3 conflicting objectives:

<table>
<thead>
<tr>
<th>package</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine</td>
<td>867</td>
</tr>
<tr>
<td>State</td>
<td>301</td>
</tr>
<tr>
<td>cp</td>
<td>154</td>
</tr>
<tr>
<td>search</td>
<td>148</td>
</tr>
<tr>
<td>examples</td>
<td>288</td>
</tr>
<tr>
<td>tests</td>
<td>1316</td>
</tr>
</tbody>
</table>