Krzysztof Kuchcinski
Krzysztof.Kuchcinski@cs.lth.se
Department of Computer Science
Lund University
Sweden
Outline

1. Introduction to CP
   - Sudoku Example

2. Examples
   - Geometrical Constraints
   - Scheduling
   - Set Constraints

3. Graph Constraints in Embedded System Design
   - Connected Components
   - (Sub-)graph Isomorphism
“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.”
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.

Jean-François Puget, ILOG/IBM

Constraint Programming is Software Engineering applied to Operations Research
A simple definition

- Constraint programming is the study of computational systems based on constraints.
- The idea of constraint programming is to solve problems by exploring constraints which must be satisfied by the solution.
Example— SUDOKU

```
2 6 3
5    7
1 4
6 5 2
4 8 1
5 9
7 3
4 1 6
```
Example– SUDOKU

Constraints programming has finally reached the masses, thousands of newspaper readers are solving their daily constraint problem.
Solution Method

Variables

\[ v[i,j] :: \{1..9\} \]
Solution Method

Variables
$v[i,j] :: \{1..9\}$

Constraints

// Rows
$v[1,1] \neq v[1,2], \ldots$

// Columns
$v[1,1] \neq v[2,1], \ldots$

// Squares
$v[1,1] \neq v[2,2], \ldots$
Solution Method

Values for first row after “simple” consistency

2 6 {1, 8..9} 3 {4..5, 7..9} {5, 7, 9} {1, 5, 8..9} {5, 8..9} {5, 8..9}
More advanced consistency

Values

2  6  \{1, 8..9\}  3  \{4..5, 7..9\}  \{5, 7, 9\}  
\{1, 5, 8..9\}  \{5, 8..9\}  \{5, 8..9\}

- values 1, 5, 8 and 9 need to be assigned to variables v[1,3], v[1,7], v[1,8], and v[1,9],
- values 1, 5, 8 and 9 can be removed from other variables in this row.
More advanced consistency

### Values

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>{1, 8..9}</td>
<td>3</td>
<td>{4..5, 7..9}</td>
<td>{5, 7, 9}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{1, 5, 8..9}</td>
<td>{5, 8..9}</td>
<td>{5, 8..9}</td>
<td></td>
</tr>
</tbody>
</table>

- values 1, 5, 8 and 9 need to be assigned to variables v[1,3], v[1,7], v[1,8], and v[1,9],
- values 1, 5, 8 and 9 can be removed from other variables in this row.

### New values

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>{1, 8..9}</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>{1, 5, 8..9}</td>
<td>{5, 8..9}</td>
<td>{5, 8..9}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution

```
2 6 9 3 4 7 1 8 5
5 4 1 8 9 6 7 3 2
8 7 3 2 5 1 6 4 9
6 1 8 5 7 3 2 9 4
3 2 5 4 1 9 8 6 7
7 9 4 6 2 8 3 5 1
1 5 6 9 3 2 4 7 8
4 8 7 1 6 5 9 2 3
9 3 2 7 8 4 5 1 6
```
SUDOKU in MiniZinc

include "globals.mzn";

array [1..9, 1..9] of var 1..9: v;

predicate row_diff(int: r) =
    all_different ([v[r, c] | c in 1..9]);

predicate col_diff(int: c) =
    all_different ([v[r, c] | r in 1..9]);

predicate subgrid_diff(int: r, int: c) =
    all_different ([v[r + i, c + j] | i, j in 0..2]);

constraint forall (r in 1..9) (row_diff(r));
constraint forall (c in 1..9) (col_diff(c));
constraint forall (r, c in {1, 4, 7}) (subgrid_diff(r, c));

solve satisfy;

output ["v = ", show(v), "\n"];

v = [| 2, 6, _, 3, _, _, _, _, _
     | 5, _, _, _, _, 7, _, _
     | _, _, _, _, _, 1, _, 4, _
     | 6, _, _, 5, _, _, 2, _, _
     | _, _, _, _, _, _, 2, _, _
     | _, _, 4, _, _, 8, _, _, _
     | _, _, 5, _, _, _, _, 1
     | _, _, 9, _, _, _, _, _
     | _, _, 7, _, _, _, _, _
     | _, _, 4, _, 1, 6 |];
Another SUDOKU example

\[
\begin{array}{cccc}
6 & 8 & 4 & 1 \\
8 & 5 & 3 & \\
2 & 6 & 8 & 9 \\
7 & & 9 & \\
5 & 1 & 6 & 2 \\
4 & 6 & 1 & \\
3 & 2 & 7 & 6 \\
3 & & & 9 \\
\end{array}
\]
Another SUDOKU example

\[
\begin{array}{ccc}
6 & 8 & 4 \\
8 & 5 & 3 \\
2 & 6 & 8 \\
7 & 9 & \\
5 & 1 & 6 \\
4 & 6 & 1 \\
3 & 2 & 7 \\
\end{array}
\]

\[
\begin{array}{ccc}
& 1 & 7 \\
\end{array}
\]

\[
\begin{array}{ccc}
8 & 5 & \\
9 & 4 & 7 \\
\end{array}
\]

\[
\begin{array}{ccc}
7 & 9 & \\
5 & 1 & 6 \\
4 & 6 & 1 \\
3 & 2 & 7 \\
\end{array}
\]

\[
\begin{array}{ccc}
& 1 & \\
\end{array}
\]

\[
\begin{array}{ccc}
& 4 & \\
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]

\[
\begin{array}{ccc}
\end{array}
\]
Pure JaCoP model

: Store store = new Store();

IntVar[][] elements = new IntVar[noRows * noColumns][noRows * noColumns];

// Creating variables.
for (int i = 0; i < noRows * noColumns; i++)
    for (int j = 0; j < noRows * noColumns; j++)
        if (description[i][j] == 0)
            elements[i][j] = new IntVar(store, "f" + i + j, 1, 9);
        else
            elements[i][j] = new IntVar(store, "f" + i + j, description[i][j], description[i][j]);

// Creating constraints for rows.
for (int i = 0; i < 9; i++)
    store.impose(new Alldistinct(elements[i]));

// Creating constraints for columns.

// Creating constraints for blocks.


Constraint programming summary

- Declarative programming style
- Different domains (finite, set, interval, real, etc.)
- Primitive and global constraints (encapsulate complex pruning algorithms)
- Program has two parts:
  - variable and constraints
  - search
- Complete and heuristic search methods
Global Constraints- Diff2

diff2([[x_1, y_1, dx_1, dy_1], [x_2, y_2, dx_2, dy_2]], ...)
include "diff2.mzn";  

int: n;                     % number squares  
int: size;                  % size of the square to fill with n squares  
array[1..n] of int: squares; % dimension of the squares  

array[1..n] of var 0..size: x; % x position of squares  
array[1..n] of var 0..size: y; % y position of squares  

constraint  
    forall(i in 1..n) (  
        x[i] <= size - squares[i] /
        y[i] <= size - squares[i] ) /
        diff2(x, y, squares, squares);  

solve :: seq_search([  
    int_search(x, smallest, indomain_min, complete),  
    int_search(y, smallest, indomain_min, complete)  
]) satisfy;  

output[show(x),show(y)];  

n=21;  
size=112;  
squares = [2,4,6,7,8,9,11,15,16,17,18,19,24,25,27,29,33,35,37,42,50];
Result

Example 0

Run time: 380 ms
Scheduling

Jobshop scheduling

- Set of \( n \) jobs and \( m \) machines,
- Each job \( i \) has \( m \) ordered tasks
- Each task has constant duration and a machine for its execution,
- Find a shortest schedule.
Cumulative constraint

\[ \text{cumulative}([s_1, \ldots s_n], [d_1, \ldots d_n], [r_1, \ldots r_n], \text{Limit}) \]
Jobshop model

```plaintext
include "globals.mzn";

int: n;
int: m;
array [1..n, 1..2*m] of int: job;
array [1..n, 1..m] of var 0..1000: t :: is_output;
array [1..n] of int: one = [ 1 | i in 1..n];
var 0..1000: end :: is_output;

constraint
  forall(i in 1..n, j in 1..m-1) ( t[i,j] + job[i, 2*j] <= t[i, j+1] );

constraint
  forall(k in 1..m) ( let 
    { 
      array[1..n] of int: d = [ job[i, 2*j] | i in 1..n, j in 1..m where job[i,2*j-1] = k-1],
      array[1..n] of var 0..1000: s = [ t[i,j] | i in 1..n, j in 1..m where job[i,2*j-1] = k-1] 
    } in 
    cumulative(s, d, one, 1) 
  );

constraint
  end = max(i in 1..n) ( t[i, m] + job[i, 2*m] );

solve :: int_search([ t[i,j] | i in 1..n, j in 1..m], smallest, indomain_min, complete) minimize end;
```
Jobshop Scheduling

Gantt diagram for job-shop scheduling.
Set Constraints

- Set variables, e.g. \( s :: \{\}..\{1..5}\) 
- Set operations, such as intersection, union, etc.
Set Constraints

- Set variables, e.g. $s \::= \{{}\..\{1..5\}\}$
- Set operations, such as intersection, union, etc.

Social golfers problem

A club has a number of golfers that play rounds in groups (the number of golfers is a multiple of the number of groups). Each round, a golfer plays with a group of different people, such that the same pair of golfers never play together twice.
Constraint Programming in Embedded System Design

Social golfers model

```plaintext

include "globals.mzn";

int: n_groups; % The number of groups.
int: n_per_group; % The size of each group.
int: n_rounds; % The number of rounds.
int: n_golfers = n_groups * n_per_group;

set of int: groups = 1..n_groups;
set of int: group = 1..n_per_group;
set of int: rounds = 1..n_rounds;
set of int: golfers = 1..n_golfers;

array [rounds, groups] of var set of golfers: round_group_golfers :: is_output;

constraint % Each group has to have the right size.
forall (r in rounds, g in groups) ( card(round_group_golfers[r, g]) = n_per_group );

constraint % Each group in each round has to be disjoint.
forall (r in rounds) ( all_disjoint (g in groups) (round_group_golfers[r, g]) );

constraint % Each pair may play together at most once.
forall (a, b in golfers where a < b) ( sum (r in rounds, g in groups) ( bool2int({a, b} subset round_group_golfers[r, g]) ) <= 1 );

solve satisfy;
```

Kris Kuchcinski
A Solution

Groups = 4, Size = 4, Round = 5

{1..4}, {5..8}, {9..12}, {13..16},
{1, 5, 9, 13}, {2, 6, 10, 14}, {3, 7, 11, 15}, {4, 8, 12, 16},
{1, 6, 11, 16}, {2, 5, 12, 15}, {3, 8..9, 14}, {4, 7, 10, 13},
{1, 7, 12, 14}, {2, 8, 11, 13}, {3, 5, 10, 16}, {4, 6, 9, 15},
{1, 8, 10, 15}, {2, 7, 9, 16}, {3, 6, 12..13}, {4..5, 11, 14}
Project

Constraint Programming in Embedded System Design

void fir(const int x[], const int h[], int y[]) {
    int i, j, sum;
    for (j = 0; j < 100; j=j+1) {
        sum = 0;
        for (i = 0; i < 8; i=i+1)
            sum += x[i + j] * h[i];
        sum = sum >> 15;
        y[j] = sum;
    }
}

/* Sample C code */

void fir(const int x[], const int h[], int y[]) {
    int i, j, sum;
    for (j = 0; j < 100; j=j+1) {
        sum = 0;
        for (i = 0; i < 8; i=i+1)
            sum += x[i + j] * h[i];
        sum = sum >> 15;
        y[j] = sum;
    }
}
Idea: Compilation Principle

DSP Data Path

\[\text{in} \xrightarrow{x} \text{ACC} \xrightarrow{+} \text{out}\]
Idea: Compilation Principle

DSP Data Path

DFG “Pattern” of MAC

Kris Kuchcinski
Idea: Compilation Principle

DSP Data Path

DFG “Pattern” of MAC

DSP Instruction Set
Idea: Compilation Principle

DSP Data Path

DFG "Pattern" of MAC

DFG of the FIR filter

DSP Instruction Set
Idea: Compilation Principle

DSP Data Path

DFG “Pattern” of MAC

DSP Instruction Set

DFG of the FIR filter

Covered DFG
Design Flow

**Step 1**
- Application Graph
- Constraints
  - Clock = 25MHz
  - Critical path = 7 nodes
- Pattern Generator
- Set of Patterns

**Step 2**
- Architecture Model
- Scheduling Constraints
  - Parallel execution
  - Deadline = 5 cycles
- Exploration Model Generation
- CP Specification

**Step 3**
- Solver
- Selected Set of Patterns
- Binding Information
- Scheduling Information
Idea: Pattern Generation Principle

Generate patterns for all $n_s \in N_{ApplicationGraph}$

- Connected component for node $n_s$
- Additional constraints
  - number of inputs $\leq$ PatternInputs
  - number of outputs $\leq$ PatternOutputs
  - critical path $\leq$ PatternCriticalPath
  - number of nodes $\leq$ PatternNumberOfNodes
  - power consumption $\leq$ PatternPowerConsumption
  - ...

- Reduce number of patterns (remove isomorphic patterns, add most frequent patterns, etc.)
(Sub-)graph isomorphism

GraphMatch($Graph_{Target}$, $Graph_{Pattern}$)
Scheduling Example - FIR filter
Scheduling Example - FIR filter
Scheduling Example - FIR filter
## Results

**Table:** Results obtained for sequential scheduling and parallel scheduling.

| Benchmarks               | | | Sequential scheduling | | | Parallel scheduling (model A) | | | Parallel scheduling (model B) | | |
|--------------------------|---|---|------------------------|---|---|------------------------|---|---|------------------------|---|
|                          | | | Cycles | Selected Cycles | Speedup | Runtime (s) | Cycles | Parallel memory accesses | Selected Cycles | Speedup | Runtime (s) | Cycles | Number cells | Speedup S1 | Runtime (s) | Percent of S1 |
| JPEG IDCT                | | | 200   | 254   | 1.3   | 18.9 | 28   | 22   | 13   | 78   | 8 | 11.55 | 14.2 | 28 | 34 | 5 | 7.47 | 10.2 | 64% |
| BF                       | | | 330   | 340   | 2.09  | 0.1  | 7    | 81   | 3    | 152  | 4 | 4.2  | 13.2 | 7  | 98  | 4  | 3.47 | 0.01 | 82% |
| MESA invert matrix       | | with mem. access ≤ 10 | 278   | 334   | 2.26  | 3.8  | 9    | 25   | 38   | 134  | 8 | 13.36 | 0.6  | 9  | 78  | 4  | 4.28 | 0.5  | 32% |
| MCRYPT cast128           | | | 424   | 464   | 1.83  | 50.5 | 18   | 202  | 8    | 155  | 6 | 2.3  | 6.6  | 18 | 219 | 3  | 2.12 | 15.9 | 92% |
| GSMenc                   | | | 387   | 433   | 3.7   | 0.1  | 9    | 16   | 15   | 132  | 8 | 27.06 | 6.0  | 9  | 68  | 2  | 6.37 | 7.2  | 23% |
| POLARSSL aes             | | | 1350  | 1658  | 2.28  | 6.2  | 15   | 280  | 27   | 739  | 8 | 5.92 | 20.2 | 10 | 482 | 5  | 3.44 | 43.6 | 58% |
| POLARSSL des3            | | | 398   | 530   | 1.76  | 5.2  | 26   | 95   | 6    | 156  | 5 | 5.58 | 23.9 | 25 | 109 | 4  | 4.86 | 24.8 | 87% |
| Maximal Speedup          | | | | | 3.7  | 27.06 | 7.47 |
| Average Speedup          | | | 2.17  | 9.98  | 4.57 |

Kris Kuchcinski 30(32)
Other constraints

- Global constraints (element, regular, cumulative, geost, gcc, knapsack, extensional_support, etc.)
- More set constraints
- Complex search strategies
- To try more http://www.jacop.eu
  (MiniZinc http://www.g12.cs.mu.oz.au/minizinc/)
Other constraints

- Global constraints (element, regular, cumulative, geost, gcc, knapsack, extensional_support, etc.)
- More set constraints
- Complex search strategies
- To try more http://www.jacop.eu
  (MiniZinc http://www.g12.cs.mu.oz.au/minizinc/)

Powered by Jacop
K. Kuchcinski.
Constraints-driven scheduling and resource assignment.

K. Martin, Ch. Wolinski, K. Kuchcinski, A. Floch, and F. Charot.
Constraint-driven instructions selection and application scheduling in the *DURASE* system.

Ch. Wolinski and K. Kuchcinski.
Automatic selection of application-specific reconfigurable processor extensions.
In *Proc. Design Automation and Test in Europe*, Munich, Germany, March 10-14, 2008.

Christophe Wolinski, Krzysztof Kuchcinski, and Erwan Raffin.
Automatic design of application-specific reconfigurable processor extensions with UPaK synthesis kernel.