EDAN01

Introduction to Constraint Programming
http://cs.lth.se/EDAN01/

Krzysztof Kuchcinski
Krzysztof.Kuchcinski@cs.lth.se

Dept. of Computer Science
LTH

October 31, 2018

Programming with Constraints

"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."

Eugene C. Freuder
CONSTRAINTS, April 1997

Quotations (cont'd)

"Constraint programming is one of the most exciting developments in programming languages of the last decade. Based on the ***strong theoretical foundation****, it is attracting widespread commercial interest and is now becoming the method of choice for modeling many types of* ***optimization problems****, in particular, those involving* ***heterogeneous constraints and combinatorial search****. Not surprisingly, therefore, constraints programming has recently been identified by the ACM as one of the strategic directions in computing research."*

Kim Mariott and Peter J. Stuckey
Programming with Constraints: An Introduction

Course organization – lectures (preliminary)

Date	Time	Room	Subject
06-11-18	10-12	M:B	Introduction and course organization.
07-11-18	13-15	D:C stora hörsalen	Introduction to JaCoP.
13-11-18	10-12	M:B	Constraints– basic notions.
14-11-18	13-15	D:C stora hörsalen	MiniZinc language.
20-11-18	10-12	M:B	Modeling with constraints.
21-11-18	13-15	D:C stora hörsalen	Modeling with constraints.
27-11-18	10-12	M:B	Finite domain constraints.
28-11-18	13-15	D:C stora hörsalen	Constraint programming and search.
04-12-18	10-12	M:B	Constraint programming and search.
05-12-18	13-15	D:C stora hörsalen	Advanced issues.

Course organization – cont'd

- Obligatory labs on constraint programming assignment in finite domain.
  - Lab instruction on the course Web site.
  - User's Guide, library, compendium, data files, examples at `/usr/local/cs/EDAN01` and `http://www.jacop.eu`
- JaCoP constraint programming library in Java (including MiniZinc) is used for the assignments (see separate User's Guide).
- Course assignments have deadlines.
- Programs need to be send to **`edan01 (at) cs.lth.se`**.
  subject: "student's id" and {Uppgift 1 | Uppgift 2 | Uppgift 3 | Uppgift 4 | Uppgift 5 }
- Course credit points – 7.5 points.
- Possibility for better grade on examination.

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

Helmut Simonis, Cork Constraint Computation Centre, University College Cork

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

Solution Method

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

Variables

```
v[i][j] :: {1..9}
```

Constraints

```
// Rows
v[0][0] ≠ v[0][1], ...
// Columns
v[0][0] ≠ v[1][0], ...
// Squares
v[0][0] ≠ v[1][1], ...
```


Solution Method

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

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[0][2], v[0][6], v[0][7], and v[0][8],
- values 1, 5, 8 and 9 can be removed from other variables in this row.

New values

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


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: sq;

constraint
   forall (r in 1..9)         (all_different ([sq[r, c] | c in 1..9]));
constraint
   forall (c in 1..9)         (all_different ([sq[r, c] | r in 1..9]));
constraint
   forall (r, c in {1, 4, 7}) (all_different ([sq[r + i, c + j] | i, j in 0..2]));

solve satisfy;

output [ "sudoku:\n\n" ] ++
        [ show(sq[i,j]) ++ if j mod 3 = 0 then if j = 9 then if i mod 3 = 0 then "\n\n" else "\n" endif
              else "  " endif else " " endif | i,j in 1..9];

sq =
 [| _, _, _, _, _, _, _, _, _
  | _, 6, 8, 4, _, 1, _, 7, _
  | _, _, _, _, 8, 5, _, 3, _
  | _, 2, 6, 8, _, 9, _, 4, _
  | _, _, 7, _, _, _, 9, _, _
  | _, 5, _, 1, _, 6, 3, 2, _
  | _, 4, _, 6, 1, _, _, _, _
  | _, 3, _, 2, _, 7, 6, 9, _
  | _, _, _, _, _, _, _, _, _
 |];
```

Standard all_different

```
%-----------------------------------------------------------------------------%
% Constrains the array of objects 'x' to be all different.
%-----------------------------------------------------------------------------%

predicate all_different_int(array[int] of var int: x) =
    forall(i,j in index_set(x) where i < j) ( x[i] != x[j] );
```


Different constraint systems

- Real/rational constraints– CLP(R), CLP(Q)
  - CLP(R), Gecode, JaCoP, SICStus, CHIP
- Finite domains constraints– CLP(FD)
  - JaCoP, Choco, Gecode, Or-tools, ECL^iPS^e, SICStus, CHIP
- Boolean constraints– CLP(B)
  - JaCoP, Gecode, Or-tools, SICStus, CHIP
- Set constraints
  - JaCoP, Choco, Gecode
- Interval constraints– CLP(I)
  - CLP(BNR), Numerica, Prolog IV

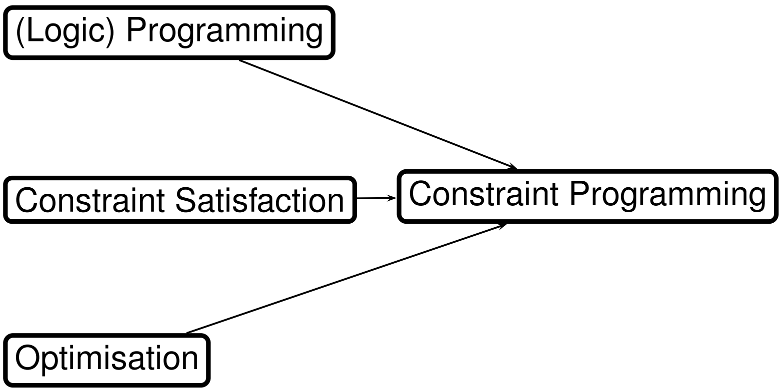
Course constraint system

We use mainly *finite domain* constraints on this course!

Constraint properties

- *may specify partial information* – need not uniquely specify the values of its variables,
- *non-directional* – typically one can infer a constraint on each present variable,
- *declarative* – specify relationship, not a procedure to enforce this relationship,
- *additive* – order of imposing constraints does not matter,
- *rarely independent* – typically they share variables.

The programming paradigm



Constraint satisfaction/solving

- Set of variables and constraints which limit the values that can be assigned to the constraint variables.
- Constraint consistency and propagation methods.
- "Encapsulation" of specific knowledge from mathematics, geometry, graph theory and operational research.

Optimization

- Finding a solution which satisfies constraints and minimizes/maximizes cost function
- Different types
  - combinatorial optimization of discrete (finite domain) variables
  - linear optimization for continuous variables
  - non-linear optimization

Examples

- Finite Domain constraints
  - SENDMORY
  - Scheduling and binding in high-level synthesis using finite domain constraints
  - Traveling salesperson problem

SENDMORY - CLP(FD)

```
  S E N D
+ M O R E
-----------
M O N E Y
```


SENDMORY - CLP(FD)

```
var 0..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 0..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;
array[1..8] of var int : fd = [S,E,N,D,M,O,R,Y];

constraint
  all_different(fd) /\
            1000*S + 100*E + 10*N + D  +
            1000*M + 100*O + 10*R + E  =
  10000*M + 1000*O + 100*N + 10*E + Y
  /\ S > 0 /\ M > 0;
```

SENDMORY (cont'd)

```
S = 9, E :: 4..7, N :: 5..8, D :: 2..8,
M = 1, O = 0, R :: 2..8, Y :: 2..8.
```


How does it work?

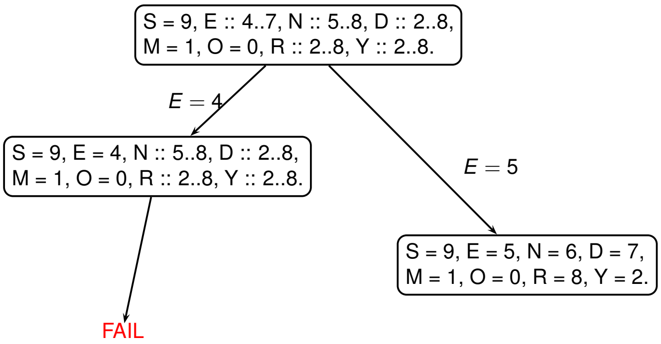
Constraint consistency

- Examples:
  - $S \neq 0 \Rightarrow S :: 1..9$,
  - if $S = 9$ then alldifferent removes 9 from all other variables domains,
  - $N :: 5..8 \Rightarrow 10 \times N :: 50..80$,
  - $D :: 2..8, N :: 5..8, (10 \times N + D) :: 52..88$,
  - etc.

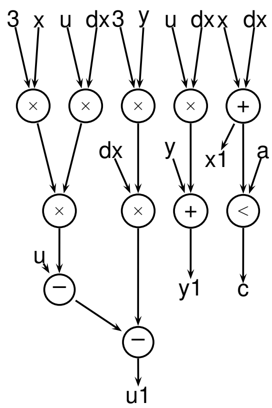
SENDMOREY (cont'd)

```
. . .
array[1..8] of var int : fd = [S,E,N,D,M,O,R,Y];

constraint
  all_different(fd) /\
            1000*S + 100*E + 10*N + D  +
            1000*M + 100*O + 10*R + E  =
  10000*M + 1000*O + 100*N + 10*E + Y
  /\ S > 0 /\ M > 0;

solve :: int_search(fd, input_order, indomain_min, complete)
    satisfy;


S = 9, E = 5, N = 6, D = 7, M = 1, O = 0,
R = 8, Y = 2.
```

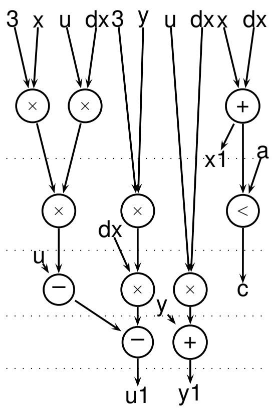
How does search work?



High-level synthesis CLP(FD)

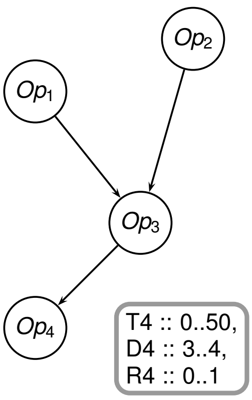


data-flow graph



scheduled data-flow graph

High-level synthesis (cont'd)

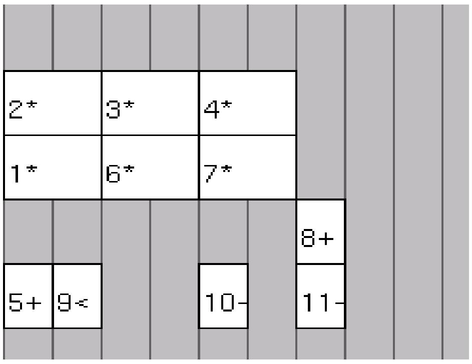


Constraints

$T1 + D1 \leq T3 \wedge$
$T2 + D2 \leq T3 \wedge$
$T3 + D3 \leq T4 \wedge$

$(T1 + D1 \leq T2 \vee T2 + D2 \leq T1 \vee R1 \neq R2)$

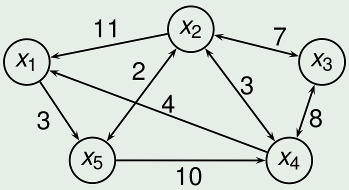
High-level synthesis (cont'd)



Scheduled design with an optimal solution

Traveling salesperson problem

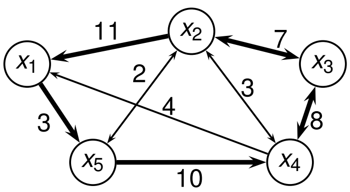
Example



$x_1 = 5, x_2 :: \{1, 3, 4, 5\}, x_3 = \{2, 4\}, x_4 :: \{1, 2, 3\}, x_5 :: \{2, 4\}.$

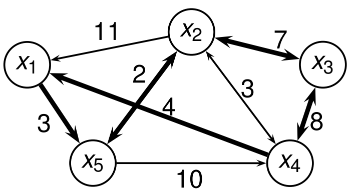
Traveling salesperson problem– model

circuit(x) $\wedge$
$d_1 = 3$ $\wedge$
element(x_2, [11, 0, 7, 3, 2], d_2) $\wedge$
element(x_3, [0, 7, 0, 8, 0], d_3) $\wedge$
element(x_4, [4, 3, 8, 0, 0], d_4) $\wedge$
element(x_5, [0, 2, 0, 10, 0], d_5) $\wedge$
$distance = \sum d_i$



Result:

```
x1 = 5; x2 = 1; x3 = 2; x4 = 3; x5 = 4;
distance = 39;
----------
x1 = 5; x2 = 3; x3 = 4; x4 = 1; x5 = 2;
distance = 24;
----------
==========
```



Methods for constraint solving

- Constraints over reals
  - Gauss-Jordan elimination,
  - simplex.
- Boolean constraints
  - SAT methods
  - Binary Decisions Diagrams (BDD's).
- Finite domain solvers
  - arc, node and path consistency methods,
  - constraint propagation (forward checking, look-ahead),
  - branch-and-bound.
- Solvers over intervals
  - interval narrowing, box consistency,
  - Gauss-Seidel elimination, interval Newton method.

CP limitations

- Many addressed problems in the area of FD constraints are NP-complete (combinatorial problems, such as scheduling, traveling salesman)
  - search for (optimal) solution has exponential complexity in general case,
  - constraints and their propagation methods try to reduce the search space,
  - possible heuristic search methods.
- Boolean satisfaction (SAT) problem is NP-complete.
- Linear programming has polynomial type solving algorithms (but in many practical cases it has long execution times).

Algorithm Complexity

- O-notation – describes asymptotic upper bound

$$O(g(n)) = \{f(n) : \text{there exist positive constant } c \text{ and } n_0 \text{ such that}$$
$$0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0\}$$

- we write $f(n) = O(g(n))$ to indicate that function $f(n)$ is a member of $O(g(n))$
- the asymptotic upper bound need to be tight; i.e.,
  - the bound $2n^2 = O(n^2)$ is tight but
  - the bound $2n = O(n^2)$ is not tight

Algorithm Complexity (cont'd)

- polynomial complexity – $O(n^k)$

$$p(n) = \sum_{i=1}^{k} a_i n^i$$

- exponential complexity – $O(a^n)$

$$p(n) = a^n$$

- comparison polynomial vs. exponential

$$\lim_{n \to \infty} \frac{n^b}{a^n} = 0 \qquad \text{for } \; a > 1$$

Algorithm Complexity (cont'd)

- An algorithm is *tractable* if it has polynomial complexity.
- *NP-complete* problems are believed to have worst-case exponential complexity solving algorithms.
- A potential solution to NP-problem can be verified in polynomial time.
- *NP-hard* problems are at least as difficult as NP-complete problems.
- There is no know method to verify a solution for NP-hard problem in polynomial time.

Constraint satisfaction problems are known to be NP-complete :-(

Applications areas

- Logistics.
- Transportation.
- Flight traffic scheduling and airport scheduling.
- Different types of planning and scheduling, e.g., nurse scheduling
- ...

Web resources

- **JaCoP v4.6 (dev)**
  `http://www.jacop.org`
  `http://sourceforge.net/projects/jacop-solver/`
  `https://github.com/radsz/jacop`
  Maven repository
- MiniZinc language
  `http://www.minizinc.org`

Questions?

Kris Kuchcinski	course responsible
Alfred Åkesson	course assistant
Noric Couderc	course assistant

e-mail:

Krzysztof.Kuchcinski@cs.lth.se
kris@cs.lth.se

course WWW page:
`http://cs.lth.se/EDAN01`