Constraint Modeling

with MiniZinc

Jakub Bulín

Department of Algebra @ CU Prague

Table of contents

In this class

- What is constraint modeling
- Basic ideas for solving
- MiniZinc: a high-level solver-independent modeling language
- Gecode: an open source yet competitive constraint solver
- Examples: how to model problems in MiniZinc
- Homework: try it yourself

Intro & the theory part

What is it

Constraint modeling (aka constraint programming, constraint optimization) is a generalization of

- Linear programming,
- Convex optimization,
- Integer programming,
- SAT solving.

A constraint model consists of decision variables (each with a domain) and a list of constraints

The goal: satisfy constraints / find all solutions / maximize a given objective function

Advantages: Close to real-life problems, easy to model, can exploit structure (lost when translated to SAT)

An example: Sudoku

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

What are the decision variables, their domains, constraints?

How to solve constraint models

- Propagation – use constraints "actively" to infer other, implicit constraints
  - forbid local assignments which lead to a contradiction
  - Arc Consistency, Path Consistency, …
- Search – try constructing a solution, modify variables which fail some constraints
  - backtracking (DFS), backjumping, …
  - for optimization: hill climbing, branch & bound, …
- Global constraints – make use of the model's combinatorial structure (which is lost when translated to SAT)
  - run specialized algorithms for parts of the model
  - different solving strategies for different families of constraints (arithmetics, LP, Boolean, ordering, scheduling, packing, circuit, DFA…)

Gecode

generic
constraint
development
environment

1886 classes, 291202 loc, 87574 lod

An overview of MinZinc

MiniZinc: (Towards a) Standard CP Modelling Language (CP'2007)

> *"A standard language for modelling CP problems will encourage ex- perimentation with and comparisons between different solvers. Although MiniZinc is not perfect—no standard modelling language will be—we believe its simplicity, expressiveness, and ease of implementation make it a practical choice for a standard language."*

Continuous development and improvement. Many mainstream Constraint solvers now understand MiniZinc.

Data types and operators

Primitive types

- boolean variables: `bool: x; bool y = true;`
- integers: `int: n; int: m = 3;`
- floating point: `float: z = 2.5;`
- strings: only for formatting the output

Operators

- arithmetic: `+, -, *, div, mod, abs(), pow(),`…
- relational: `=, !=, <, <=,` …
- logical: /\\, \\/, `->, <->, not,`…

Data structures

Arrays, e.g. `array[0..100,0..100] of var float: temp;`

- array comprehension (similar to Python):
  `[i + j | i, j in 1..3 where j < i]`
  evaluates to
  `[1 + 2, 1 + 3, 2 + 3]`
  which is
  `[3, 4, 5]`
- aggregation functions: `min, max, sum, product, forall, exists`

Sets, e.g. `set of int: values = {1, 4, 9, 16};`

- set comprehension (similar)
- set functions: `in, subset, superset, union, inter, diff, symdiff`

A constraint model

Each MiniZinc model has these parts (but the order of statements does not matter)

- declare parameters, e.g. `int n;`
  - before execution each parameter will have a fixed value
  - separate model from data (command-line arguments or a .dzn file)
- declare decision variables, e.g. `array[1..10] of var bool: x;`
  - the solver will choose values for these
- define constraints, e.g. `constraint x+y > z;`
- solve statement: `solve satisfy;` or `solve maximize/minimize <obj-function>;`
- (optional) `output <array-of-strings>;`

Examples of constraint models

A few examples

- Sudoku
- Math Software homework: The Unruly riddle
- Math Software homework: Schedule of classes
- Groups on n elements
- Finite projective planes
- RC4
- Graph coloring
- SAT

Finite projective planes

$N^2 + N + 1$ points,
$N^2 + N + 1$ lines,
$N + 1$ points on each line,
$N + 1$ lines through each point,
any two points lie on exactly one line,
any two lines intersect at exactly 1 point,
there are four points which do not lie on one line

A finite projective plane exists if $N = p^k$. Is this if and only if? $N = 6$ ruled out by theory, $N = 10$ by massive computation, $N = 12$ open.

Stream cipher RC4 – seed generation

input a key $k_1, \ldots, k_{2^n} \in 2^n$
output a seed $\pi \in \mathrm{Sym}_{2^n}$

$$a_{-1} = 0$$
$$\pi_{-1} = \mathrm{id}$$

for $i = 0$ **to** $2^n - 1$ **do**

$$a_i = a_{i-1} + \pi_{i-1}(i) + k_i \pmod{2^n}$$
$$\pi_i = (i\ a_i) \circ \pi_{i-1}$$

return π_{2^n-1}

Goal: compute the key from a known seed.
Idea: Model the whole computation process.

Learn more

Advanced topics

- PyMzn – MiniZinc Python wrapper
  - convert between MiniZinc and Python data objects
  - useful tools for input and output processing
  - solver specific commands
  - `pymzn.minizinc('test.mzn', 'data1.dzn', parallel=4, output_mode='dict')`
- Assertions – data consistency testing
  - `constraint assert(ammount >= 0, "Invalid input");`
- Search annotations – prescribe solving strategy
  - branch on `my_array`, variables with smallest domain first, largest values first
  - `solve :: int_search(my_arrray, first_fail, indomain_max, complete) satisfy;`
- Predicates – define your own (analogous to functions)
  - ```
    predicate even(var int:x) =
        let { var int: y } in x = 2 * y;
    ```
    ```

Want to know more?

- MiniZinc website, a PDF tutorial! `http://www.minizinc.org/`
- Two Coursera courses `http://www.coursera.org/`
    - Basic modeling for discrete optimization
    - Advanced modeling for discrete optimization
- Hakan Kjellerstrand's MiniZinc webpage with lots of example models `http://www.hakank.org/minizinc/`
- PyMzn homepage `http://paolodragone.com/pymzn/`
- Gecode homepage `http://www.gecode.org/`
- Three courses at MatFyz
    - NOPT042 Constraint programming
    - NMAG563 Introduction to complexity of CSP
    - NMMB536 Optimization and Approximation CSP