

# Dominion

A constraint solver synthesiser

Lars Kotthoff

`larsko@cs.st-andrews.ac.uk`

# Motivation

- ▷ there are many different constraint solvers (e.g. Minion, Gecode, Choco, Eclipse. . . ), all of which have different design decisions
- ▷ these decisions affect the performance of the respective solver
- ▷ sometimes these differences determine whether a problem can be solved in a reasonable amount of time or not
- ▷ there are many more decisions which affect solver performance

## Examples

- ▷ management of backtrack memory
  - ▷ trailing
  - ▷ copying
  - ▷ recomputation
- ▷ variable types
- ▷ propagation queue
- ▷ consistency levels
- ▷ search order

more thorough investigation in CIRCA preprint

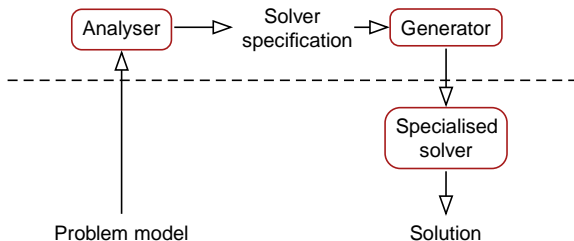
“Constraint solvers: An empirical evaluation of design decisions”

# The Dominion system

## Idea

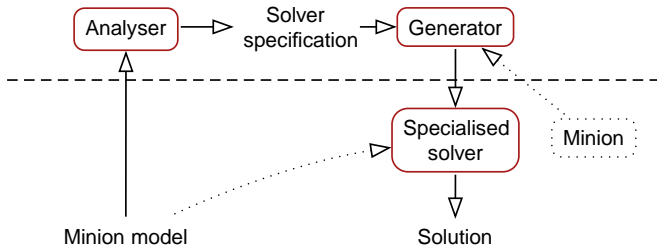
- ▷ synthesise a constraint solver for a problem class (e.g.  $n$ -Queens) or instance (e.g. 3-Queens)
- ▷ make the design decisions most suitable for this problem
- ▷ use the best algorithms and data structures
- performance improvements while still finding the correct solutions
  
- ▷ foundations for a £1 million project at the Constraint Programming group at the University of St Andrews

## Architecture



- ▷ **Analyser** fixes design decisions based on analysis of the problem in the **Solver specification**
- ▷ **Generator** synthesises a solver according to the specification

Work so far



- ▷ preliminary implementation of the Dominion system in Haskell based on Minion
- ▷ **Analyser** parses Minion input files
- ▷ **Generator** modifies the Minion source code to create a specialised solver

## Modifications done by the generator

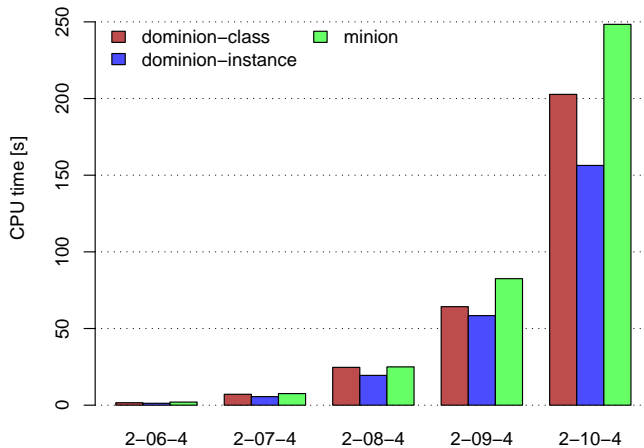
- ▷ include only the constraints and variable types needed
  - ▷ generate new types for variables with integer domains
  - ▷ remove infrastructure code which is not required for the problem
- 
- increases speed of the solver
  - speeds up solver compilation by several orders of magnitude

modified Minion source can be used in two ways –

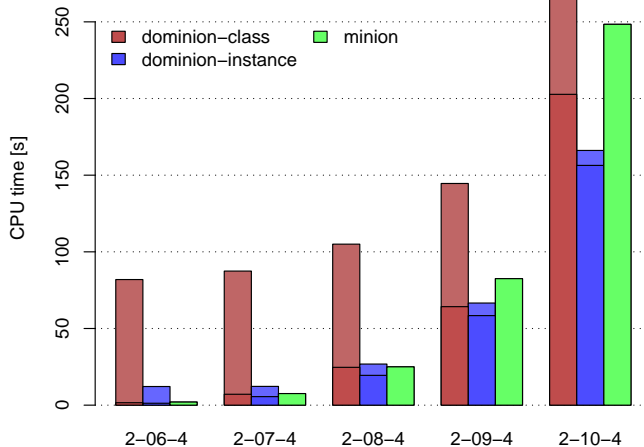
**class solver** compile to Minion which is able to solve problem models of the same **problem class** as the analysed model

**instance solver** compile a Dominion-generated source file to a solver which is only able to solve the model of the analysed **problem instance**, but is specialised to a higher degree

# Benchmarking



Social Golfers – solve times



Social Golfers – solve times + generation times

- ▷ similar improvements for other problems
- ▷ initial results are very promising
- ▷ impressive performance improvements over standard Minion considering the small modifications
- ▷ for large problems, Dominion wins even when taking the time to run Dominion and compile the specialised solver into account

Future work

## Shortcomings

- ▷ only modifies Minion, does not synthesise a solver from scratch
- ▷ analysis of the Minion input file very limited
- ▷ no true problem class solver

### Make it faster!

- ▷ switch between more fundamental design decisions
- ▷ what heuristics are appropriate for a particular problem class/instance?
- ▷ how to determine what level of consistency to enforce?
- ▷ what specialised algorithms and data structures can we generate for a particular problem class/instance?

Questions?

How does Dominion know that the solver it generates is any good?

- ▷ the final system will have several ways of deciding which implementation decisions to make
  - ▷ heuristics in the code
  - ▷ a tuner that empirically decides which implementation is best by running several on the problem
  - ▷ offer a selection of algorithms in the specialised solver to be chosen at run time
  - ▷ manual annotation of the Solver specification

Why not have a general solver that has all the different implementations and choose at run time?

- ▷ there are some things that cannot be done in a general solver
  - ▷ specialised variable types which encode the domain
  - ▷ specifying a problem to solve only by giving the values for problem class parameters
- ▷ including more algorithms and data structures increases compile time and size of the binary exponentially
- ▷ depending on the number of different implementations to choose from, switching at run time could be expensive