Oxiflex - A Constraint Programming Solver for MiniZinc written in Rust

Bachelor's thesis

Faculty Artificial Intelligence of the University of Basel
Department of Mathematics and Computer Science
Research Group
https://ai.dmi.unibas.ch/

Examiner: Examiner Prof. Dr. Malte Helmert
Supervisor: Supervisor Simon Dold)

Gianluca Klimmer
gianluca.klimmer@stud.unibas.ch
2019-915-594

Hand-In-Date

Abstract

This thesis discusses the thesis template using some examples of the Turing Machine.

Table of Contents

1

Introduction

2

Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSP) are mathematical questions defined as a finite set of variables whose value must satisfy a number of constraints or limitations. When solely talking about the problem without the algorithmic finding of a solution, these are called Constraint Networks.

Example:

$$w = \{1, 2, 3, 4\}$$
$$y = \{1, 2, 3, 4\}$$
$$x = \{1, 2, 3\}$$
$$z = \{1, 2, 3\}$$

where:

$$w = 2 * x$$
$$w < z$$
$$y > z$$

We define variables w, y, x and z. Variables w and y can both have one value from $\{1, 2, 3, 4\}$ and variables x and z can have a value from $\{1, 2, 3\}$. The constraints then restrict which values are valid from their respective domains. Here $w = 2 \times x$ restrict the value of x to be double of w for example. If there are no constraints for variables, the constraints are still there but they allow every assignment. These constraints are called trivial constraints and are usually omitted.

In this example we define constraints in a mathmatical notation. There are no formal restrictions on stating constraints neither by their complexity nor by the number of variables involved. To make it easier to reason about, we model constraints as binary constraint sets. Constraints are then sets of valid value pairs for two specific variables. Instead of stating the desired relation between any variables, we list all valid value pair tuples in a set. Constraint $w < z$ then becomes $(R_{wz} = \{(1, 2), (1, 3), (2, 3)\}$ which contains all possible value pairs for the two varaibles w and z.

We define Constraint Networks formally:

A (binary) constraint network is a 3-tuple $C =< V, \mathrm{dom}, (R_{uv}) >$ such that:

- V is a non-empty and finite set of variables,
- dom is a function that assigns a non-empty and finite domain to each variable $v \in V$, and
- $(R_{uv})_{u,v \in V, u \neq v}$ is a family of binary relations (constraints) over V where for all $u \neq v : R_{uv} \subseteq \mathrm{dom}(u) \times \mathrm{dom}(v)$

And we define our example formally:

$C =< V, \mathrm{dom}, (R_{uv}) >$ with

- variables:
  $V = \{w, x, y, z\}$
- domains:
  $\mathrm{dom}(w) = \mathrm{dom}(y) = \{1, 2, 3, 4\}$
  $\mathrm{dom}(x) = \mathrm{dom}(z) = \{1, 2, 3\}$
- constraints:
  $R_{wx} = \{(2, 1), (4, 2)\}$
  $R_{wz} = \{(1, 2), (1, 3), (2, 3)\}$
  $R_{yz} = \{(2, 1), (3, 1), (3, 2), (4, 1), (4, 2), (4, 3)\}$

The goal in CSP is then to find a Assignment that satisfies all constraints. For this simple example a possible assignment would be $(w \mapsto 2), (x \mapsto 1), (y \mapsto 4), (z \mapsto 3)$.

2.1 MiniZinc

MiniZinc [1] is a free and open-source constraint modeling language developed at and by Monash University in Australia. It allows us to express Constraint Satisfaction Problems in a mathmatical notation-like way.

MiniZinc example

```
var 1..4: w;
var 1..4: y;
var 1..3: x;
var 1..3: z;

constraint w = 2 × x;
constraint w < z;
constraint y > z;

solve satisfy;
```

MiniZinc is only the language to express a problem domain. Once a problem domain is specified in MiniZinc we can give the problem to multiple solvers to solve it each. In this way we can compare the performance of various solvers on the same problem domain. MiniZinc Domain files have the file extension `.mzn`.

MiniZinc also provides a way to parametrize a problem domain. This is a great way to scale a problem space up and see how increasing the problem space affects the solving speed. A great example for this is the Queens Problem (See Section 2.2). We define the Queens Problem domain once and can then run specific problem instances for different n. This makes it really easy to compare the solving speed for the queens problem when $n = 8$, $n = 16$ or $n = 32$. Files where we specify parameters for MiniZinc files are called data files and have the extension `.dzn`. We can than combine `.mzn` files with `.dzn` files to created FlatZinc files.

2.1.1 FlatZinc

FlatZinc is a simpler problem specification language provided by the MiniZinc package. It is designed to be used by solvers directly. MiniZinc files are translated to FlatZinc files in a pre-solving step. FlatZinc files have the file extension `.fzn`.

Translating from MiniZinc to FlatZinc maps more advanced instructions from MiniZinc to primitives supported in FlatZinc. An analogy to this translation is compiling a C program to Assembly where MiniZinc is C and FlatZinc is Assembly. FlatZinc therefore requires solvers to support a set of standard contraints called "FlatZinc builtins". Builtins need to be implemented to be a fully compatible FlatZinc solver.

Simple example translated to FlatZinc

```
array [1..2] of int: x_introduced_2_ = [1,-2];
array [1..2] of int: x_introduced_3_ = [1,-1];
array [1..2] of int: x_introduced_4_ = [-1,1];
var 2..4: w:: output_var:: is_defined_var;
var 1..4: y:: output_var;
var 1..3: x:: output_var;
var 1..3: z:: output_var;
constraint int_lin_eq(x_introduced_2_,[w,x],0):: defines_var(w);
constraint int_lin_le(x_introduced_3_,[w,z],-1);
constraint int_lin_le(x_introduced_4_,[y,z],-1);
solve satisfy;
```

The translation of the variable declations is straight forward. But our constraints are each split into two parts and translated into linear combinations. For example is constraint $w = 2 \times x$ translated to the builtin called `int_lin_eq` which is defined as follows:

int_lin_eq builtin

```
predicate int_lin_eq(array [int] of int: as,
array [int] of var int: bs,
int: c)
```

With the restiction on given parameters to the constraint.

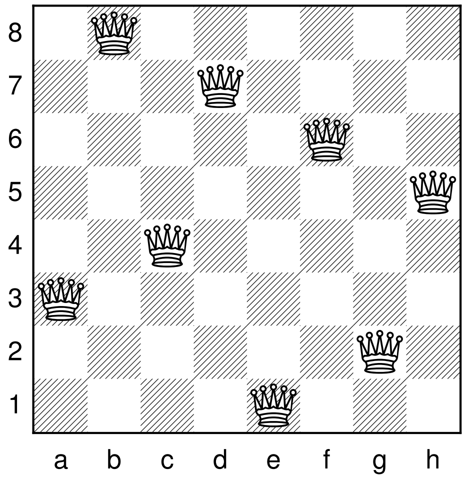
int_lin_eq builtin constraint

$$c = \sum_i \text{as}[i] \times \text{bs}[i] \tag{2.1}$$

Note that MiniZinc already does some basic level of inference. The FlatZinc variable w can only have values between 2 and 4 in the translated FlatZinc file. But in the MiniZinc file we defined w with the domain $\{1, 2, 3, 4\}$. This means MiniZinc infers that w can not be value 1 and removes it from its domain declaration. Due to the constraint $w = 2 \times x$, the variable w has to be double of x and x must have at least value 1. Therefore excluding 1 as possible value for w.

2.2 Queens Problem

Also called the Eight Queens Puzzle, the Queens Problem is an example of a classic constraint satisfaction problem that involves placing eight queens on an 8x8 chessboard in such a way that no two queens threaten each other. That is, no two queens can share the same row, column, or diagonal.



The Eigth Queens Puzzle is really good suited as an example problem domain for constraint satisfaction problems because it is easy to understand and can also easily be scaled up to increase complexity for a solver. By generalizing the problem from a fixed 8×8 grid size to an $n \times n$ grid with n queens, the problem remains the same in principle, but gets way harder to solve.

MiniZinc Model for N-Queens Problem

```
int: n;

array [1..n] of var 1..n: q;

predicate
noattack(int: i, int: j, var int: qi, var int: qj) =
qi != qj /\
qi + i != qj + j /\
qi - i != qj - j;

constraint
forall (i in 1..n, j in i+1..n) (
noattack(i, j, q[i], q[j])
);

solve satisfy;
```

This MiniZinc model defines an array of variables q where each index corresponds to a column on the chessboard and the value at each index represents the row position of the queen in that column. The constraints ensure that no two queens are on the same row, column or diagonal.
This model is from a past MiniZinc challenge by Reza Rafeh July 2005 and Peter Stuckey September 30 2006. See https://github.com/MiniZinc/minizinc-benchmarks/blob/master/

queens/queens.mzn for reference.

3 Oxiflex

As part of this thesis we present oxiflex, a minimal constraint satisfaction problem solver from scratch for MiniZinc written in rust. Oxiflex is a FlatZinc solver that can be used as an backend to MiniZinc. This means oxiflex supports the minimal requirements for a solver to take advantage of the MiniZinc toolchain.
Oxiflex is open-source and available under the MIT license under https://github.com/glklimmer/oxiflex.

3.1 Rust

3.2 Dependecies

3.2.1 flatzinc

A FlatZinc parser for rust.

3.2.2 structopt

3.2.3 hyperfine

3.3 Solver

3.3.1 Naive Backtracking

3.4 Inference

3.4.1 Forward Checking

3.4.2 Arc Consistency

4 Results

Results, Graphs and stuff.

5 Conclusion

Time for some interpretation.

Bibliography

[1] N. Nethercote, P.J. Stuckey, R. Becket, S. Brand, G.J. Duck, and G. Tack. Minizinc: Towards a standard cp modelling language. In C. Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.

A

Appendix

University of Basel

Faculty of Science

Declaration on Scientific Integrity
(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis:

Name Assessor: ______________________________

Name Student: ______________________________

Matriculation No.: ______________________________

I attest with my signature that I have written this work independently and without outside help. I also attest that the information concerning the sources used in this work is true and complete in every respect. All sources that have been quoted or paraphrased have been marked accordingly.

Additionally, I affirm that any text passages written with the help of AI-supported technology are marked as such, including a reference to the AI-supported program used.
This paper may be checked for plagiarism and use of AI-supported technology using the appropriate software. I understand that unethical conduct may lead to a grade of 1 or “fail” or expulsion from the study program.

Place, Date: ______________ Student: ______________

Will this work, or parts of it, be published?

No

Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of: ______________________________

Place, Date: ______________ Student: ______________

Place, Date: ______________ Assessor: ______________

Please enclose a completed and signed copy of this declaration in your Bachelor’s or Master’s thesis.