Test Generation

Bilal Syed Hussain

University of St Andrews

1 Purpose

The aim is to generate Essence specifications for testing purposes.

1.1 Correctness

After generating a specification, either an error occurs upon refinement, or eprime(s) models are produced. These eprime(s) are converted to minion input using Savilerow and solved using Minion. If the specification has solutions these are converted back to essence and validated against the original specification.

1.2 Things taking too long

If the generated specification produces no models in a reasonable amount of time, then either the specification is too hard or we need better refinement rules.

2 Validation

2.1 Validate Solution

This would be put be lettings from the essence solution back into the original specification, type checking and then checking if the constraints hold.

2.2 Backtrack Solver

For very small specification, A simple backtrack solver could be written which could be used for validation.

2.3 Sampling

Instead of generating a simple model, using the compact heuristic, we sample a set of models. The simplest way to do this would be to generate a random model until we have n models (making sure to remove duplicates). If model were inconsistent i.e. some models produced a solutions while other models did not then there is a bug in one of the refinement rules.

2.4 Other Solvers

Since Savilerow can output constraint models for multiple solvers (e.g minion, gecode, domination, minizinc) we could see if we get the same set of solutions from each solver. If the number of solutions from each solver is different then it might indicates that there is a bug. If one solver does not produce a solution, but other solvers do, then there was either a bug during refinement or a bug in one of the solvers.

3 Errors

Some errors are not useful, Savilerow outputting it can't parse a number because it too large.

4 Domain Generation

When generating domains we can't just generate then randomly because we most likely get a specification that would never produce a single model in a reasonable amount of time. Ideally we would like to able to get coverage of the whole specification space.

To achieve coverage we generate domains with different levels of nesting, example of different levels of nesting:

0. Ints, booleans, enums, unnamed types.
1. – Set of int
   – mset of int
   – relation of int * int
   – partition from int
   – function from int –> int
   – matrix indexed by int of int
   – (int, int)
2. – set of set of int
   – function set of int -> set of int

When we generate domains with n levels of nesting, we have already tested domains which have levels of nesting less then n hence it possible to generate all specification given enough time.

4.1 Enums & Unnamed types

Unnamed types, can't be validated at the moment, this will have to be fixed

4.2 Attributes

4.3 Ints are complex

Even ints are not simple, example included:

- int
- int$(1, 2, 4)$
- int$(2..3)$
- int$(1, 2..5, 9)$ union int$(1, 2, 3)$
- int$(1..0)$
- int$(1, 3, 2)$

Edge cases include empty domains, domains which are out of order and operations on these domains.

4.4 Dependent domains

Domains can be depend on other domains (these get mostly inlined during refinement). More complex domains can depend on given instances.

5 Constraint Generation

Since we already have the generated domains, we generate constraints on these domains. This would solve the problem of having undefined variables.