Forge 1

Due: 11:59PM Friday, February 7, 2020

Stencil Files

We have provided three stencil files for the three problems. They can be found in `/course/cs1950y/pub/forge1`

Installing Forge

To install Forge, you'll first need DrRacket, which you can download here. If you have an existing installation, please make sure you are running version 7.4 or greater. To install Forge, go to `File > Install Package`. Type `forge` as the package source and then Install (or Update). To tell DrRacket you are writing in Forge, add the line `#lang forge` to the top of every Forge file you use.

Please make sure to save your file before running anything. Also, if highlighting and indentation is not working properly, try retyping `forge` in the `#lang forge` line to load the highlighting. This might be necessary when you open an existing file.

Note that we are actively working to make Forge better. If you run into a potential bug, please add an issue on Github.

Problem 1: Binary Relations

The following Forge model constrains a binary relation to have a collection of standard properties:

```
sig Atom { -- define a unary relation `Atom`
  r: set Atom  -- define a binary relation `r` which is `Atom -> Atom`
}
run {
  some r                -- nonempty
  r.r in r              -- transitive
  no iden & r           -- irreflexive
  ~r in r               -- symmetric
  ~r.r in iden          -- functional
  r.~r in iden          -- injective/one-to-one
  Atom in r.Atom        -- total
  Atom in Atom.r        -- surjective/onto
} for 4 Atom
```

Part 1: Forge Syntax

Each constraint in the run statement describes the property in the comment. For example, `some r` forces the relation `r` to contain at least one tuple, and hence be nonempty. However, some of the constraints are a bit trickier. For example, why does `r.~r in iden` constrain `r` to be injective? For each of the properties other than the nonempty property, write a brief (around 1-2 sentences) explanation of why the Forge constraint describes the property. Include these explanations as comments in the Forge file.
You can read more about some of the properties defined above here. Note that what is described as "total" above is called "left total" on Wikipedia.

Part 2: Making the Relation Satisfiable

A finite binary relation cannot have all these properties at once. Which properties, if eliminated individually, allow the remaining seven properties to be satisfied? For each such property eliminated, name the property and give an example (as a comment in the Forge file) of an instance (atom names and the relation `r`) that satisfies the rest of the properties.

You can use Forge to help you. The `run` command instructs the analyzer to search for an instance satisfying the constraints in a universe of at most 4 atoms.

To eliminate a property, just comment it out (with two hyphens in a row at the start of the line). Running should bring up a browser window that shows sample universes in which the properties specified are true. To give the example in a comment, use the following format: `Atom = {Atom0, Atom1}, r = {Atom0->Atom1, Atom1->Atom1}` (where this example is for a universe with two atoms and two edges, 0 to 1 and 1 to itself).

Problem 2: Distributivity

You may notice repeated subexpressions that can be factored out, making the overall expression more succinct. For example, the expression `p.mother.brother + p.father.brother`, denoting `p`'s uncles, can be written instead as `p.(mother + father).brother`. Simplifications like this rely on the assumption that certain algebraic identities hold, such as

- distributivity of join over union: `s.(p + q) = s.p + s.q`;
- distributivity of join over difference: `s.(p - q) = s.p - s.q`; and
- distributivity of join over intersection: `s.(p & q) = s.p & s.q`

for a given set `s` and binary relations `p` and `q`.
For each of the above three identities, in a comment, say whether it holds, and if not, give a counterexample (atom names and relations, in the same format as for Problem 1.2). Here is an example of how you might check the first using Forge:

```
pred union {
   Atom.(p + q) = Atom.p + Atom.q
}
check union for 4 Atom
```

The `check` command tells Forge to look for a counterexample within a universe of 4 elements. When you find that a property does not hold, try and obtain the smallest counterexample you can, by reducing the scope (for example, replacing `for 4` with `for 3`).

Problem 3: Directed Tree Properties

A tree has several canonical definitions within computer science that you may be familiar with. In one common definition, a tree is a data structure consisting of nodes and child pointers to other nodes. Each child has only one parent (except the root node, which has zero parents), and each node may have one or more children (note: we are not restricting to binary trees). The child pointers cannot form a cycle. This may also be referred to as a directed tree.

The above description and Wikipedia page describe directed trees informally and formally in English and mathematical notation. A directed tree can also be described as a relation that satisfies some properties. What exactly are these properties? For this problem, you should express these properties in relational logic without using quantifiers. We allow the use of the relational as in `some x` where x is relational expression, but not `some x: TYPE | ...`. Note you should express the properties for a tree, not a forest.

Additional Forge Resources

If you are struggling with Forge syntax or operators, some particularly useful resources that the TAs would recommend include:

- The official Alloy documentation
- A quick reference guide to most Forge operators
- A helpful powerpoint presentation for syntax and basic examples

Note that Forge is largely similar to Alloy, but differs in a few ways, so if you ever have a question about Forge's syntax, don't hesitate to ask at hours or on Piazza!

Handing In

Run `cs1950y_handin forge1` from a directory holding your 3 Forge files (`binary_relation.rkt`, `distributivity.rkt`, `directed_tree.rkt`). You should receive an email confirming that your handin has worked.