Advanced Programming Languages and Techniques

- Announcement: May 25, 2024
- Due: July 2, 2024

Guidelines

- Submit a single compressed file `tar.gz` file containing your solution.
- You can use Haskell's standard library if you want.

Assignment 3

In this assignment you will implement a rule engine in Haskell that can infer new *facts* from a set of given *rules* and some initial facts. The rule engine should be able to evaluate the rules and derive new facts iteratively until no more new facts can be derived. A rule engine is realized as a function

```
infer :: Rules -> Facts -> Facts
```

that takes the set of rules and the facts that considered to be true and return all the facts that can be produced (initial and inferred). The types of `Facts` and `Rules` are deliberately not completely specified and left to you to define them as you wish.

In the following, two versions of the rule engine of different expressivity are described that you should implement.

1. A simple rule engine assumes that a `Fact` is simply a `String`. A `Rule` $C_1, \ldots, C_n \to A$ is a list of conditions C_i (essentially facts) and when all conditions are true then a new fact A is derived. The newly inferred fact A will be used in the next iteration to decide whether another rule with A in its list of conditions should be fired.

   Consider the following self-explanatory set of rules for a smart home system:

   1. `["It's morning",  "Temperature is below 18°C" ] --> "Turn on heating"`
   2. `["Temperature is below 18°C", "Turn on heating"] --> "Check for open windows"`
   3. `["No one is home", "Turn on heating"] --> "Send notification to homeowner"`

   Given the initial facts

   - `"It's morning"`

- `"Temperature is below 18°C"`

the engine should infer the fact `"Turn on heating"` from the first rule. This new fact will be added to the already known facts. In the next iteration, the second rule will be triggered since both `"Turn on heating"` and `"Temperature is below 18°C"` are true. As a result the `"Check for open windows"` will be added. In the next iteration no new facts will be added and the inference process terminates.

2. A more expressive rule engine assumes that a condition may contain variables. For this to work a fact should be a list of `String`s and a condition is a list of either `String`s or variables. For example, consider the rule

```
[ ["Room", (Var X)],
  ["MotionDetected", (Var X)] ] --> ["TurnOnLights", (Var X)]
```

which essentially expresses the rule: "if motion is detected in any room, then turn on the lights for that room". A rule fires if the engine can substitute the variables with values and there are facts that match the conditions. For the sake of the example assume that the initial facts are `["Room", "LivingRoom"]`, `["Room", "BedRoom1"]`, `["Room", "BedRoom2"]` and `["MotionDetected", "BedRoom1"]`.

The aforementioned rule fires only for `X = "BedRoom1"` and therefore the inferred fact is `["TurnOnLights", "BedRoom1"]`. In order for this to work, the only rules that are allowed should be the rules that all the variables appearing in the rule should also appear in the conditions of the rule. For example, the rule

```
[ ["Room", (Var X)],
  ["IsMorning"] ] --> ["SetTemperature", (Var X), (Var T)]
```

because the variable `T` does not occur in the conditions of the rule.