Forge Lab 1: Graphs

Part 1

Begin with the following Forge signature describing `Cat`s, each of which has a set of `friends`:

```
sig Cat {
  friends : set Cat
}
run {}
```

As crucial as the friendship of kittens is, we're using it here as a playful framing of very real problems in computer science.

Averting *Cat*astrophe

Run this Forge model and view several instances in the visualizer by clicking *Next*. This model has a few unpleasant characteristics that we would like to constrain:

No lonely kittens

Make sure that all Cats have at least one friend. Using the predicate below as a template, replace `...` with your implementation:

```
pred NoLonelyKittens {
  ...
}
```

Cats are friends with *other* kittens

Some of the cats are claiming themselves as friends! Introduce a fact ensuring that the friends of a cat do not include itself. Using the predicate below as a template, replace `...` with your implementation:

```
predicate OutsideFriends {
  ...
}
```

Friendship isn't a one-way street

Well, *Friendship* might be a one-way street in Providence, but these cats aren't from around here. Visualize an instance of your model so far, and note that the arrows connecting instances of `Cat` may be one-directional. Introduce a predicate `TwoWayStreet` ensuring that if one cat considers another cat as a friend, the other cat reciprocates the friendship:

```
pred TwoWayStreet {
  ...
}
```

You can combine these all together into one predicate:

```
pred friendship {
      NoLonelyKittens
      OutsideFriends
      TwoWayStreet
}
```

Try running this predicate with `run friendship` to see all the cats!

Kitty Bacon

There's one special `Cat` in our universe, an actor by the name of Kitty Bacon! Add this signature to your model:

```
one sig KittyBacon extends Cat { connectionsOf : set Cat }
```

If you view an instance of this model, you should see exactly one `Cat` with the name `KittyBacon`.

Degrees of Kitty Bacon

Kitty Bacon is very connected. All kitties are in his connections:

```
pred ConnectedKittyBacon {
    connectionsOfKittyBacon
    Cat - KittyBacon = KittyBacon.connectionsOf
}
```

`ConnectedKittyBacon` depends on the relation `KittyBacon.connectionsOf`, which should consist of the union of KittyBacon's friends of friends (not including KittyBacon themself) and KittyBacon's friends. Complete the definition:

```
pred connectionsOfKittyBacon{
    friendship
    KittyBacon.connectionsOf = ...
}
run {ConnectedKittyBacon and connectionsOfKittyBacon}
```

Look at the visualization of your instance. Are *all* cats connected to `KittyBacon` via their network of friends? If you got *UNSAT*, double check that your definition of `KittyBacon.connectionsOf` isn't including too many cats.

A common pattern in Forge is to test a suspicion by phrasing it differently in a second predicate, and checking if both predicates are equivalent. An alternative way to express connected-via-friends is the *transitive closure operator*, `^`. `KittyBacon.^friends` includes friends, friends of friends, friends of friends of friends, and so on.

```
pred SuperConnected {
    connectionsOfKittyBacon
    Cat - KittyBacon in KittyBacon.^friends
}
```

And two predicates are equivalent iff one implies the other. Fill in the predicate below:

```
pred ConnectedKittyBacon_equals_SuperConnected {
  ...
}
```

Finally, check the predicate:

```
check ConnectedKittyBacon_equals_SuperConnected for exactly 3 Cat
```

Were any counter examples found? What happens if you increase the scope of the model to include an additional cat?

```
check ConnectedKittyBacon_equals_SuperConnected for exactly 4 Cat
```

Modify KittyBacon.`connectionsOf` to include their `friends of friends of friends` and check for counterexamples again. If none are found, increase the scope of the `check` to `exactly 5 Cat`. Is it possible to modify KittyBacon.`connectionsOf` without using `*` or `^` so that for any number of degrees of separation, and for any number of cats, no counterexample to this predicate can be found? If not, why? Write your response in a comment.

Part 2

At this point, `ConnectedKittyBacon_equals_SuperConnected` is true for all 4-Cat instances, but not for all 5-Cat (and larger) instances. That is, `ConnectedKittyBacon_equals_SuperConnected` is true for some instances and false for others. Based on your response to Part 1, let's investigate what circumstances make it false. Please copy and paste the following definitions into the end of your Forge lab file:

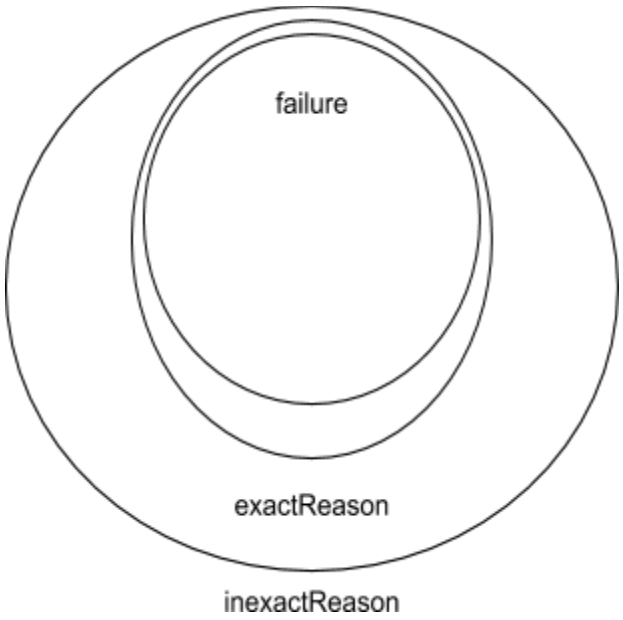
```
pred failure {not ConnectedKittyBacon_equals_SuperConnected}

pred inexactReason { some c:Cat-KittyBacon | c not in
KittyBacon.connectionsOf }

run { failure and not inexactReason } for exactly 5 Cat

pred exactReason { /* FILL */ }

check { connectionsOfKittyBacon implies (failure iff exactReason) } for
exactly 5 Cat
```



This picture shows the relationships between the sets of instances that satisfy each predicate: the ovals indicate sets of instances. Our starting guess, `inexactReason`, just says that `ConnectedKittyBacon` fails. This isn't yet specific enough to truly capture the situations when `ConnectedKittyBacon` is not equal to `SuperConnected`. Your job is to find a better explanation: one that is "tighter" than `inexactReason`. Fortunately, because `inexactReason`

contains all the cases where the property fails, we can ask Forge to help us blame portions of our spec. When we ask it for instances where {`failure` and not `inexactReason`} it says there are no such instances: the run is *unsatisfiable*.

See if you can fill in `exactReason` with a more specific explanation. If your reason is exact, `validateExactReason` should return no instances. If there are counterexamples, feel free to use these to refine your hypothesis.

Note that there may be many valid `exactReason` predicates. Please pick the one that you believe is best.

Do not be discouraged if you are having trouble figuring out an exact reason. You will receive **full credit** for an honest attempt!

Optional: Just for Fun

Throughout the final portions of the lab, you may have noticed that `KittyBacon` is never included in `KittyBacon.connectionsOf` in any instance. What circumstances make it always false?

Check-off

Call a TA over to talk about your responses.