Computational Applied Logic [CSC 503]

AQUARIUM BUILDER

Final Project

By,
Namitha Mariam George (ngeorge3)
Vishnu Radhakrishnan Nair (vradhak)

I. Overview

The realisation that predicate logic can be applied even in our ecosystem, inspired us to take a closer at our surroundings. While researching this, we encountered Ralph W. Lewis who proposed creating a system of theories for evolutionary theory [1]. This was the inspiration in taking up a project in **marine biology domain**.

This project aims to create a city aquarium and populate it with a variety of aquatic species so as to maximise the number of species taking into consideration the various inter species relationships and other external aspects like salinity level, light, temperature etc. Aquariums are quite popular in terms of aesthetic appeal and are also popular among hobbyists or even people who like fishes as pets. Hence, the project can scale to any level as per user preference.

Our main motivation for the project was the dismal realization regarding how the space and environment built to protect the aquatic life forms go under utilized due to lack of knowledge about the compatibility of species to other lifeforms and the prevailing conditions. This is just one of the ethical aspects of this. The marine biologists could also use this tool to verify their evaluations and goals while considering certain environments. We started out with the general ones and built up on them for the more specific ones. Of all the roles an aquarist must perform, requiring knowledge of every species is the most challenging and that is what our project aims to alleviate.

Key Challenges:

We had to go through a lot of research for accumulating the knowledge, finding an appropriate SAT solver and connecting it to the codebase. Coming up with the design of the system.

Design space:

The design space seems to be blank for this particular set of problems. So our project is an innovation over it.

Goals:

Initial goal :

Build an environment and return the longest list of species that could coexist in that environment.

Achieved result:

1. Built an environment and longest list of species that could coexist in the environment.
2. Give the longest list of species that could exist in a given environment. (We can also set an upper limit to the number of species for the environment)
3. Generate all environments with the maximum list of species in each of them.

As we can see we achieved more than that of what we proposed during the project demo and proposal.

II. Related Work

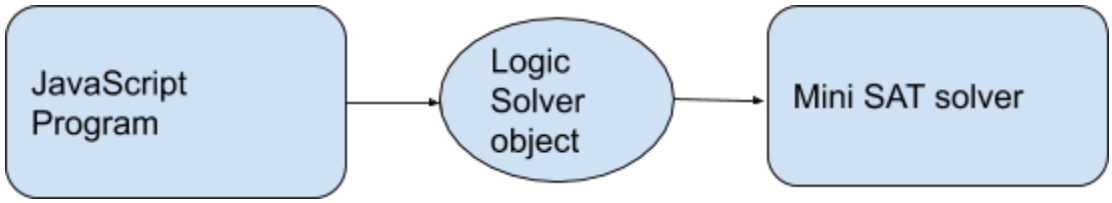
Currently, there seems to be no software targeted at building an aquarium that can address the task of deciding on the aquatic life that should be put together in an aquarium. As per our research, there is not any design or a working module which does the same. So this is a novel idea. Hence, we were presented with the need to have good prior knowledge regarding aquatic species or else a need for extensive research for aquarist beginners. Our project aims at making this an easier task by having all the constraint based data in one place which users can merely query to find out if their ideas regarding coexisting species is feasible.

What we learned in our class has a lot to do with how we will be approaching our project. The propositional logic and the conjunctive normal form helped us represent the relationship of our variables to each other. The entire aspect of the rule creating weighs totally on these concepts. What we learned in predicate logic along with how we tackle satisfiability problems in ASP helped a lot. This is why we are using a SAT solver for solving the problem.

III. Approach

We decided on using Mini SAT solver for our project, as it is reputed as an industrial level SAT solver. The language we used was Javascript. This method of connecting the solver to Javascript worked well for us.
MiniSAT solver requires its input to be in Conjunctive Normal Form (CNF).

Architecture Overview:



Eg:
So we want to create a rule like *cannotCoexist(shark, Marlin)*
In the code it will be defined like *solver.require(Logic.implies("Marlin", "-Shark"));*
This rule will be converted to CNF form (*-Marlin v -Shark*) by the Logic solver object. The CNF form will be fed into the MiniSAT solver.

In short, the program execution is handled by JavaScript while the solver is hidden in the object. Below is a use case diagram and a high level system architecture diagram.

Outputs:
We can actually run it in three ways.

1. It automatically gives the longest list of species and that corresponding environmental details from all the species that we populated in the data collection.

```
***BUILD YOUR AQUARIUM!***

Options:
1.Get fishes and required conditions for the largest possible single aquarium.

2.Input your prefered aquarium conditions and get largest list possible for it.

3.Get largest list for every possible aquarium conditions.

Select one of the above options :
1
Enter Tank limit :
100

 Longest list Conditions:
[ 'Fresh', 'Light', 'Lukewarm' ]
 => Longest list Species:
[
  'Bettas',     'Catfish',
  'CrayFish',   'Eels',
  'Gar',        'Guppies',
  'Loaches',    'Planktons',
  'Plecos',     'RainbowFish',
  'Snails',     'StickBug',
  'SwordFish',  'Tetra',
  'Trout',      'Turtle'
]
Number of species:
16
```

2. This run actually takes in the environmental conditions like the temperature , salinity and lighting levels from the user along with the maximum population size and gives the longest corresponding list as the output.

```
***BUILD YOUR AQUARIUM!***

Options:
1.Get fishes and required conditions for the largest possible single aquarium.

2.Input your prefered aquarium conditions and get largest list possible for it.

3.Get largest list for every possible aquarium conditions.

Select one of the above options :
2
Enter Tank Conditions you prefer :
Lukewarm Dark
Enter Tank limit :
20

 Longest list Conditions:
[ 'Dark', 'Fresh', 'Lukewarm' ]
 => Longest list Species:
[ 'Freshwater-Angelfish', 'Mollies', 'Platies' ]
Number of species:
3
```

3. In this mode the program gives a list of different environments and its maximum list of species contained in each of them

```
 Longest list Conditions:
[ 'Fresh', 'Light', 'Lukewarm' ]
 => Longest list Species:
[
  'Bettas',     'Catfish',
  'CrayFish',   'Eels',
  'Gar',        'Guppies',
  'Loaches',    'Planktons',
  'Plecos',     'RainbowFish',
  'Snails',     'StickBug',
  'SwordFish',  'Tetra',
  'Trout',      'Turtle'
]
Number of species:
16

 Longest list Conditions:
[ 'Dark', 'Fresh', 'Lukewarm' ]
 => Longest list Species:
[ 'Freshwater-Angelfish', 'Mollies', 'Platies' ]
Number of species:
3

 Longest list Conditions:
[ 'Cold', 'Fresh', 'Light' ]
 => Longest list Species:
[
  'Cherry-Shrimp',
  'Hillstream-Loach',
  'Paradise-Gourami',
  'ParadiseFish',
  'Peppered-Cory',
  'Rhinogobius-duospilus',
  'RosyBarb',
  'Zebra-Danios'
]
Number of species:
8
```

IV. False starts, pitfalls, failures

When we started out on the project the biggest challenge we faced was pipelining the javascript with the Mini SAT solver[2]. We decided on this solver as it is an industrial level, well-designed and well-documented tool. The MiniSAT solver didn't directly connect in Javascript.

After a lot of research, we settled on LogicSolver object which contains a copy of the MiniSAT solver. This seamlessly integrated into our project and all our rule constructions are via this.

The next challenge faced by us was the herculean task of data collection and converting them to rules. This was due to numerous fish species and their intricate dependencies and also the lack of a centralized source from where we may collect our data. Clearly, in the actual development of such a tool, data collection from experts is an inevitable step.

Due to the large amount of data we had to work with, validating our results was another difficulty. We chose to use python but again, a project with so many dependencies is best handled by a SAT solver. Here, we were able to appreciate the use of raw formal logic in solving certain problems. We compared the outputs to the subset of the rules that we passed for that run.

V. Outcome and Evaluation

Validating the code:

The python code we initially used to test a subset of our data was difficult to scale for larger subsets and it gave only one output contrary to a SAT solver which gives all possible outputs. Setting up an ideal python code involved more work than the main project. Hence, we ran smaller subsets of the rules and validated it with the result produced by our project. Thus we were able to prove the correctness of the solution.

The *code seemed efficient* as we just created the rules and converted it to the most basic state and fed it into a SAT solver to get the solution. No complex algorithm was used. The outputs were as we had expected as we did runs with a subset of the rules. So we can say that the results were correct and stable with the rules provided.

Sample output-

These results are completely according to what we had expected and what was required as from the project proposal.

Project difficulty:
The project was a bit more difficult that we initially estimated.

1. When we started out on the project the biggest challenge we faced was pipelining the javascript with the Mini SAT solver[2]. We solved this by using a Logisolver object as a wrapper and passing the rules through this object.
2. There was too much data to be converted to rules. We spent a lot of time making all the rules and passing it into the SAT solver.
3. Passing inputs were also an issue. We ultimately passed it to the solver as assumptions through the Logic solver object.

Learnings:
One of the main learning achievements we made regarding the project was that we saw, used and got familiar with the aspects of Mini SAT solver, an industrial level SAT solver which was totally unfamiliar to us. We also learned how to connect it with Javascript (another language we were unfamiliar with).

Another way in which we increased our experience is by designing solution to our problem. We were able to come up with different models for our problem statement and compared its pros and cons and finally decide on our current solution. Thus we became more familiar with problem solving techniques through our work

Outside the syllabus and the topic we learned a lot about different saltwater and freshwater aquatic species along with their environmental requirements. We were also able to peek into different inter-species relationships shared by different species in the environment.

Grading:
Goals:
Initial goal : *as promised during project demo and project proposal.*
Build an environment and longest list of species that could coexist in that environment.
Achieved result:

1. Built an environment and longest list of species that could coexist in the environment.
2. Give the longest list of species that can exist in a given environment. (We can also set an upper limit to the number of species for the environment)
3. Generate all environments with the maximum list of species in each of them.

As you can see we already achieved our goals perfectly and was able to do two other extra features as well. So we think Full 100 marks + some extra credit can be awarded for this project.

VI. References

[1] RW Lewis. Evolution: A system of theories. Perspectives in Biology and Medicine, 23:551–572, 1980.

[2] http://minisat.se/

[3] Our own project proposal and midway report

[4] https://github.com/meteor/logic-solver

[5] https://www.petcoach.co/article/friend-or-foe-freshwater-fish-compatibility-for-a-happy-tank/

[6] http://www.theaquariumwiki.com/wiki/Coldwater_Aquariums

[7] https://aquariumadviser.com/best-saltwater-aquarium-fish/