# Project Summary

*Short summary of the project setting.*

# Propositions

*List of the propositions used in the model, and their (English) interpretation.*

# Constraints

*List of constraint types used in the model and their (English) interpretation. You only need to provide one example for each constraint type: e.g., if you have constraints saying “cars have one colour assigned” in a car configuration setting, then you only need to show the constraints for a single car. Essentially, we want to see the pattern for all of the types of constraints, and not every constraint enumerated.*

# Model Exploration

*List all the ways that you have explored your model – not only the final version, but intermediate versions as well. See (C3) in the project description for ideas.*

## Distance -1 Bug

I encountered a bug where the following constraint led to every distance proposition being true:

A screen shot of a computer code

Description automatically generated

To diagnose things, I first added constraints saying every node reaches itself with distance 0, and nothing else can:

A computer code on a black background

Description automatically generated

This led to the theory being unsolvable. I tried printing the constraints being added, but it just looked like a mess. So, instead, I took a step back and tried to draw out the situation. The diagram ultimately led to my understanding of what was wrong, and looked like this:



My issue was with the for-loops and where I was using the n1, n2, n3 in the constraints. The revised and working code was this:

A computer screen with many colorful text

Description automatically generated with medium confidence

Interestingly, this caused the number of solutions with my first example graph to plummet from 80,190 to just 1. This makes intuitive sense, since there should be only a single solution when the graph is pre-determined.

## Strongly Connected

The idea with this property was to not use the Reachable proposition, but rather the distance ones. The reason for this is that the Reachable propositions may all be set true to satisfy those constraints – it helps to force a graph to be disconnected, but can’t be used to check it.

Consequently, this is how the strongly connected property was built:

A computer screen with text

Description automatically generated

Most interestingly for exploration of the model, the last two constraints tied together this “STRONGLY\_CONNECTED” proposition with the constraints that define the graph property. This notion led to the creation of a new type of propositions to explore things (and constrain it if I want):

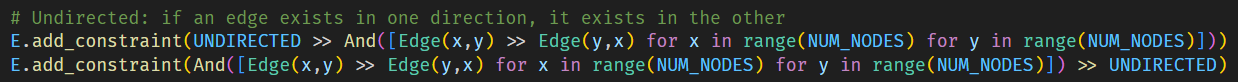
A computer screen shot of text

Description automatically generated

Also, as part of testing this property, I added a new graph that was a cycle on 4 nodes (which is strongly connected). This pattern will allow for other properties to be used as well.

## Undirected Bug

As another property, I tried to add a new proposition that indicated if a graph was undirected (every edge implies that the opposite edge exists). This was the first attempt:



…but it unfortunately didn’t fix the proposition. Every time it was run, it would randomly set to true or false, *even though it was running on a fixed graph of four nodes in a cycle.*

From this point, I looked at what the encoding object offered to print the constraints, and used “introspect”:

A black background with white text

Description automatically generated

Looked good, but still way too many constraints. Filtering just for “undirected” listed this:

A black background with many numbers

Description automatically generated with medium confidence

From this point, it was clear that the variables were named incorrectly, and I wasn’t using the right Edge objects. After fixing the edge creation (using parameters such as “f'n{x}',f'n{y}'”), it seemed to work and I tested this by re-running it several times and watching for the property. It rarely came up as a randomly generated undirected graph, and I manually confirmed this was correct from the adjacency lists.

## Exploring Likelihoods

With things in place, I was ready to start exploring the likelihood of a graph having certain properties:

A black screen with blue and orange text

Description automatically generated

A black background with white text

Description automatically generated

Interestingly, we can see that about half of the undirected graphs are completely disconnected – if they are connected, and undirected, then it should be strongly connected as well.

## { *Note From Prof* }

*The content in this section is precisely the type of things we’re looking for. It’s about 70-80% complete of what we’d expect from an A+ project, with more exploration of solutions and bugs being missing.*

# Jape Proof Ideas

*List the ideas you have to build sequents & proofs that relate to your project.*

1. “If there are a pair of nodes that aren’t adjacent, then there is no edge between them.”  
   \* If an edge exists, then it is adjacent (optionally, adjacent means edge either way)  
   ∀x.∀y.(Padj(x,y)→(PEdge(x,y) ∨PEdge(y,x))) ⊢ ∀x.∀y.(¬PAdj(x,y) → ¬PEdge(x,y))  
   ∀x.∀y.(Padj(x,y)→(PEdge(x,y) ∨PEdge(y,x))) ⊢ ∀x.∀y.( PEdge(x,y) → PAdj(x,y))

… Original attempt above &^^^^

…realized that the premises were not strong enough. Subsequent attempt:

∀x.∀y.((Padj(x,y)→(PEdge(x,y) ∨PEdge(y,x))) ∧ (PEdge(x,y) ∨PEdge(y,x))→Padj(x,y)) ⊢ ∀x.∀y.( PEdge(x,y) → PAdj(x,y))

A screenshot of a computer

Description automatically generated

1. “If a node has a self-loop, then it can always reach itself with one higher distance.”
2. “In a chain of four nodes, the distance from the first to last is three.”

# First-Order Extension

*Describe how you might extend your model to a predicate logic setting, including how both the propositions and constraints would be updated.* ***There is no need to implement this extension!***

## Predicates

We will start with the propositions used for the base model, and extend to other predicates of interest.

* Edge(x,y): There is an edge from node x to node y
* Degree(x,n): Node x has degree n
* Connection(x,y,n): Node y is the nth connection starting at node x
* Connected: Graph is connected
* Distance(x,y,n): A path exists of length n from node x to y
* *Equality* : Implicitly lets us compare two objects for equality

To handle sorts, we would need the following predicates:

* Node(x): x is a node
* Number(n): n is a whole number

## Functions

* sum(n, n’): Function that returns an object that is the sum of n and n’
* sub(n, n’): Function that returns an object that is the subtraction of n and n’ (i.e., n – n’)

## Constraints

Some of the constraints we might have:

* If an edge exists between nodes, then their distance is 1
* If a node x can reach y with distance n, and y can reach z with distance n’, x can reach z with distance n+n’  
   )
* If distance\_x\_y\_n holds, then distance\_x\_z\_(n-1) must hold for some z connected to y
* If we can reach every node with some distance, then the graph is connected
* If a graph is connected, then between every pair of nodes there is a finite distance

## Potential Sequents/Theorems

*Things you might be able to prove, if you had infinite time/resources/jape window size*

…

# Useful Notation

*Feel free to copy/paste the symbols here and remove this section before submitting.*