# Project Summary

The aim of this project is to take some initial configuration of student preferences for the courses they would like to TA, as well as the Prof preferences of the TA’s they would like to have, and then find a workable assignment of TA’s to the courses. Constraints will be needed for specific elements like dedicated TA’s, sufficient grades in the course, distribution of graduate TA’s, etc.

# Propositions

Only a small handful of propositions in this model (full project could expect several more).

1. **Assigned\_s\_c**: Represents whether a student s is assigned to a course c.
2. **StudentPref\_s\_c\_l**: Indicates a student s’ preference level l for a course c.
3. **ProfPref\_p\_s\_c\_l**: Represents a professor p's preference level l for a student s to be assigned to a particular course c.

# Constraints

Here we list several of the constraints used in the implementation.

1. **Unique Assignment Constraint:** A student cannot be assigned to more than one course.
   * **¬(Assigned(s, c1) ∧ Assigned(s, c2))** for all pairs of distinct courses **c1** and **c2**.
2. **Preference Avoidance Constraint:** A student should not be assigned to a course that they rank as their least preference.
   * **StudentPref(s, c, 1) → ¬Assigned(s, c)** for each student **s** and course **c**.
3. **Nash Equilibrium Constraint:** No two students should be in a position where they would both prefer to swap their assigned courses.
   * **Assigned(s1, c1) ∧ Assigned(s2, c2) → ¬(StudentPref(s1, c2, l2) ∧ StudentPref(s2, c1, l1))** for **l2 > l1**.
4. **Imperfect Solution Constraint**: No student is assigned to a course that is their top preference.
   * **Assigned(s, c1) → Or(StudentPref(s, c2, l2) ∧ StudentPref(s, c1, l1))** for **l2 > l1**, indicating there's always a higher preferred course **c2**.

# Model Exploration

## Missing TAs

A screen shot of a computer

Description automatically generated

Here, I noticed there were no TAs. It occurred to me that I need a constraint to say how many TAs we need for a particular course. This led me to the following constraint…

A black background with white text

Description automatically generatedA screen shot of a computer program

Description automatically generated

Unfortunately, while this assigned a TA to every course, it had two issues. (1) some had many TA’s, and (2) some had just one (see right). The fix was to identify the exact two TA’s for a course like this:

A computer screen with text and numbers

Description automatically generated

This led to exactly 2 assigned to every course.

## Better Student Pref Display

I was struggling to get a sense of what the preferences led to with respect to the course selection. To overcome this, I made the grid of students x courses prettier, and made the entries of course selections green. This let me very quickly see that every course has 2 Tas, and what their preference was.

A computer screen shot of code

Description automatically generated

## Forcing a Bad Nash

The Nash Equilibrium constraint – that no two students mutually want to swap – is a fairly complex one:

A screen shot of a computer program

Description automatically generated

In order to test that things are working, I changed it ever so slightly so that we force every pair of students assigned to different courses to be violating the Nash Equilibrium:



This led to the following solution:

A black and white grid with green numbers and white text

Description automatically generated

Indeed, we can confirm that every assignment has a better option on every other course.

## Forced Imperfection

To explore the model further, I decided to force sub-optimal preference assignments. Similar to the Nash Equilibrium, the idea is to force *some* other option for every student that looks better. This runs up again the Nash Equilibrium constraint in an interesting way, and forces a more complex solution to be computed. This was the constraint specification:

A screen shot of a computer code

Description automatically generated

Note that the options range over all courses, and not just levels on a single course. This is one solution that was computed:

A black board with white text and green numbers

Description automatically generated

Note that every student has another course of a higher preference, but no Nash Equilibrium constraint is violated.

# Jape Proof IdeasA screenshot of a computer Description automatically generated

## If a student isn’t assigned to 2 of 3 courses, they must be assigned to the third.

The way I’ll interpret this is that (1) a student must be assigned to one of three courses, and also (2) they are not assigned to c1 and not assigned to c2. Then we deduce that they are assigned to c3.

We will use propositions such as **Ps1ATc1** to mean that s1 is assigned to course c1. The final sequent we have is then,

(Ps1ATc1 ∨ Ps1ATc2 ∨ Ps1ATc3), ¬Ps1ATc1 ∧ ¬Ps1ATc2  
 ⊢ Ps1ATc3

## If a student is assigned to a course, then it isn’t one with a preference of one.

**Premise**: for all students and all courses, if the student assigns it a preference of one, then they are not assigned the course. This follows closely the exact constraint in our code:

A screen shot of a computer

Description automatically generated

∀x.∀y.(PrefOne(x,y)→¬Passigned(x,y))

**Additional premises:** We have a student i1 that is assigned to course i2:

actual i1, actual i2, Passigned(i1,i2)

**Conclusion:** They don’t have a preference of one for the course:

¬PrefOne(i1,i2)

A screenshot of a computer

Description automatically generated

## There must be two students ranked highly, if a course has two assigned

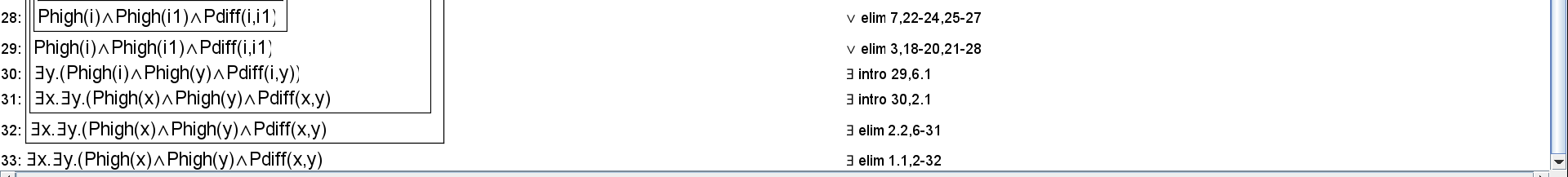
Will need the following premises to make this work:

* Course needs two (unique) students assigned  
  ∃x.∃y.(Pdiff(x,y)∧Passigned(x)∧Passigned(y))
* Can’t assign a student with “low” ranking  
  ∀x.(Plow(x) → ¬Passigned(x))
* Students are ranked either “low” or “high”  
  ∀x.(Plow(x)∨Phigh(x))

The conclusion is just that ∃x.∃y.(Phigh(x) ∧ Phigh(y) ∧ Pdiff(x,y)). The full sequent:  
∃x.∃y.(Pdiff(x,y)∧Passigned(x)∧Passigned(y)), ∀x.(Plow(x) → ¬Passigned(x)), ∀x.(Plow(x)∨Phigh(x)) ⊢ ∃x.∃y.(Phigh(x) ∧ Phigh(y) ∧ Pdiff(x,y))

A screenshot of a computer

Description automatically generated



# First-Order Extension

## Domain of Discourse

* Natural numbers (used for levels)
* Objects for profs
* Objects for students (TAs)
* Objects for courses

## Predicates

* Assigned(x,y): Student x is assigned to course y for a TA’ship
* Preference(x,y,z): Student x has a preference level of z (1->5) for course y
* ProfPref(w,x,y,z): Prof w has a preference for TA x at level z (1->5) for course y
* MaxGrad(x,y): The maximum number of graduate students TA’ing course x is y
* Assume that we have equality for objects (e.g., or )
* …

We may want to specify the types of individual objects, so that the quantification is a little more oriented to the objects we use. These would be the types for this particular project:

* Student(x): x is a student
* Prof(x): x is a prof
* Course(x): x is a course
* Num(x): x is a number

## Constraints

* All TAs can be assigned to only one course  
     
    
  If we want to make sure that the objects are of the correct type, then the formula would be:

Because it makes the formulae unwieldy to always include the types of the objects we quantify over, we will assume that they are implicitly included whenever we have a quantifier.

* No course gets a TA ranked 2 or lower by the instructor
* Some profs can veto certain TAs  
   Interpreted as: a prof can forgo assigning *any* level to a particular TA (across all courses)  
     
    
   If instead, we interpret it as having a value of 1 for the TA
* Nash equilibrium: no swap of TAs/Courses should lead to a better outcome (i.e., more preferences satisfied)

# Conclusion

This report gives you a good look at what we're aiming for in a final project for CISC/CMPE 204, but this time it's all about the TA assignment project. It digs into how to tackle assigning TAs to courses, balancing what students and profs want, and making sure all the rules are followed. For a top-tier, A+ project, you'd probably need to add more details, like really getting into the nitty-gritty of the model and how it works. But as it stands, this is a solid example of what a good project looks like.