# Project Summary

The NHL season requires a rigorous amount of planning to ensure the optimal season schedule has been created. Our project aims to create a working NHL season schedule, on a week-by-week basis, taking into account the various rules and restrictions associated with the league. Additionally, our project will focus on the altered rules of the most recent 2020-2021, COVID influenced season. Based on the NHL schedule of this season, our project follows the rules of divisions of 8 teams, each playing each other a total of 8 times.

Our model aims to create a schedule for one week, in which we can then apply to craft the resulting season schedule for 16 weeks. Using the restrictions of our constraints, our model looks to see if a proposed game is valid, and then schedules it into the weekly game schedule. Using this week-by-week process, and the restriction of constraints, we intend to create the 20 week schedule for the season.Image

# Propositions

**Game(date, Team1, Team2):** Each game played must contain a date of play, a home team, and an opposing away team. **date:** can be day 1-7. Any one of the days in the week. **Team1:** Can be any one of the 8 teams in the division of play. **Team2:** Can be anyone of the 8 teams, and not Team1, the other team in the game.

\**SideNote*: We recognize that Team2 can simply be written as Team1 on a single game basis. However, when trying to schedule the games for the entire week, we need to be able to keep track of more factors than simply the opposing team not being Team1. Hence the reason for Team2, so we can keep track of what team is a valid opposing team for Team1, based on the additional factors of our constraints.

**Team(loc, teamNum):** Each team in the division must contain if it is the home or away team, as well its team number out of the 8. **loc:** True if the team is home, False if away. **teamNum:** Can be a number from 1-8.

**Date(day):** The date of the game can either be a valid day of play, or it is a day in which the game cannot be played. **day:** True, if the game played on that day. False otherwise.

**teamNum():** The number associated with team, to represent it. **teamNum:** Can be either team 1,2,3,4…8.

# Constraints

**D = Day T = Team G = Game**

**-** Each day must contain at least one game being played

D1 D2 D3 … D7 must be true

* Each team cannot play a game three days in a row

G(Dx, T1, T2) G(Dx+1, T1, T2) G(Dx-1, T1, T2) G(Dx+2, T1, T2)

* For every game, the location cannot be home for both teams

T(TRUE, x) T(FALSE, y)

-Each team must play only 4 games a week

G(Dx, T1, Ty) G(Dy, T1, Ty) G(Dq, T1, Ty) G(Dz, T1, Ty) G(Dz+1, T1, Ty)

* Each team has to play at least one home game, and one away game each week

(G(Dx, T1, Tx) T(TRUE, x)) (G(Dx+1, T1, Tx) T(TRUE, x)) (G(Dy, T1, Tx) T(TRUE, x)) (G(Dy+1, T1, Tx) T(FALSE, x))

* Each team must play only one game a day

G(Dx, T1, Tx) G(Dx, T1, Tx)

* Each Team cannot play itself

G(Dx, T1, T1)

# Model Exploration

In exploring our model, we initially struggled on how we wanted to represent the schedule, eventually deciding on a grid system to represent it. Our grid is a 3x14 grid, representing the 7 days of a weekly schedule. Each two rows represents a day, with the even rows representing the away team, and the odd rows representing the home team. This can be seen in our initial sketch for the gird below:

Image  
We are still struggling to create a fully working model for our project, yet we do have a direction in which we are aiming to achieve. We essentially want to initially create a working model for one week of scheduling. Our thought process is to use this process of modelling based on the grid, to be able to successfully model one week. We believe that once we can create a working model for one week, we can then apply this process to fulfil our goal of scheduling a 16 week schedule. Our idea of representing the schedule in the grid, has been the most promising outlet in our modelling of the project.

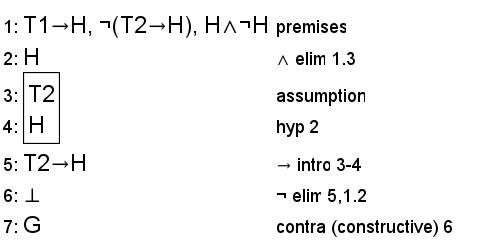
Our largest struggle in creating our model is our lack of understanding of the syntax related Bahaus and Dsharp. We believe that our model would not be too hard to fully create, yet we are struggling to grasp how to represent it using the syntax associated with Bahaus. In our model we have tried to code our constraints as best as possible, yet we are unsure if they are fully proper as we are struggling to grasp how to properly code them using Bahaus. With feedback and clarification, the team believes we will be able to create our working model.

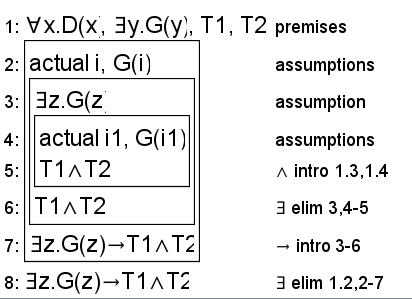
\**Side Note:* We have more constraints, that are not yet implemented in the code, as they apply to the schedule, for multiple weeks, rather than one singular week of scheduling. As we work out one week’s schedule, and begin to expand it to a full season, we will implement these constraints in our code.

# Jape Proof Ideas

**D = Day T = Team G = Game H = Home**

For our Jape proofs, we decided to model small instances of our larger project. Because our project is focused on scheduling games for a season, we decided to focus our Jape proofs on small instances that ensured the validity of games.

Our first Jape proof models that a game is valid, when one team is home, and the opposing team is away. As seen below:

Our next proof had to incorporate some first order extension in order to properly model the instance. In this instance we are looking to prove that there is a game on each day of the week. Our Jape proof models that for all days in the week, there exists some game, and that implies that that game is played between Team 1 and Team 2. Every day has game

Our final sequent (not yet proven), is to prove that each team cannot play three games three days in a row. Looking at our sequent, if a game is played on day 1 and day 2, that implies that no game is played on day 3, or the day before day 1. Therefore, if we have a game on day 1 and day 3, we can see that there is no game on day 2.

# Image

# Requested Feedback

Are our constraints on the right track in their use of Bahaus syntax? If not, which constraints need to be updated?

Is there a way in which our second Jape proof can be written not using first order extension?

We are looking for more clarity on the use/importance of DSharp in the main section. Is there need to implement more functions in our code in order to ensure DSharp performs its required task?

# First-Order Extension

To extend our project

# Useful Notation

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