Missouri University of Science & Technology — Department of Computer Science
Fall 2022 — **CS 5408: Game Theory for Computing**

Homework 4b: Dynamic Games - Extensive Game Solvers

Instructor: *Sid Nadendla* — **Due:** *December 9, 2022*

Instructions: Students who did not follow any of the following instructions will be ignored and a zero grade will be rewarded accordingly.

- The main goal of this assignment is to implement a Python package `gtclab` for representing and solving both normal and extensive games from scratch.
- You are **not** allowed to import any other Python library, other than the ones that are already imported in the code-base.
- You are also **not** allowed to add, move, or remove any files, or even modify their names.
- You are also **not** allowed to change the signature (list of input attributes) of each function.

Problem 0 Extensive Game Representation *1 pts.*

Copy all the code you wrote for HW-2b, that supports representation of games in extensive form, into their respective locations in HW-4b. These files include:

- hw2b/gtc-lab/base/state.py ⇒ hw4b/gtc-lab/base/state.py
- hw2b/gtc-lab/base/tree.py ⇒ hw4b/gtc-lab/base/tree.py
- hw2b/gtc-lab/models/extensivegame.py ⇒ hw4b/gtc-lab/models/extensivegame.py

Problem 1 Subgame Perfect Equilibrium *4 pts.*

Implement each of the following classes and methods listed below, which can be found in hw4b/gtc-lab/solvers/spne.py:

- `spne()`: This class solves any perfect-information extensive game using the notion of *subgame perfect equilibrium*. In this algorithm, the value of a given state is the Therefore, define the following four static methods accordingly:
  - `set_state_value()`: Set the value of the state, whose label is given.
  - `get_state_value()`: Get the value of the state, whose label is given.
  - `set_subtree_equilibrium()`: Set the equilibrium of the subtree rooted at the state, whose label is given.

- `get_subtree_equilibrium()`: Get the equilibrium of the subtree rooted at the state, whose label is given.
- `find_subtree_NE()`: Find the Nash equilibrium of the subtree rooted at the state, whose label is given.

Problem 2 Perfect Bayesian Equilibrium *5 pts.*

This solver is designed to solve for perfect Bayesian equilibrium in ***two-player signaling games*** with two signaling strategies at the sender with two types, and an arbitrary number of strategies at the receiver.

Implement each of the following classes and methods listed below, which can be found in hw1/gtc-lab/solvers/spne.py:

(a) `pbe()`: This class solves any perfect-information Bayesian extensive games using *perfect Bayesian equilibrium*. Kindly implement the algorithm using the following three methods, as discussed below:

- `is_2player_signaling_2x2sender()`: This method checks if a given game is a two-player signaling game with a sender with two types and two signaling strategies.
- `is_sequentially_rational()`: This method checks if a given behavioral strategy satisfies sequential rationality for every player.
- `compute_consistent_beliefs()`: This method returns an assessment pair by computing consistent beliefs for a given behavioral strategy. Note that if the computed belief is undefined, an appropriate distortion should be introduced to find a consistent belief and evaluate how this belief converges as distortion converges to zero.

Problem 3 Validation *3 pts.*

(a) **Subgame Perfect Equilibrium:** Retrieve the Jupyter notebook from HW 2b with the implementation of the extensive game shown in Figure 1. Solve this game using your own `spne` solver.

(b) **Perfect Bayesian Equilibrium:** Write a Python script in Jupyter notebook that uses your implementation of `gtclab` package to compute *perfect Bayesian equilibrium* for the extensive game shown in Figure (b).

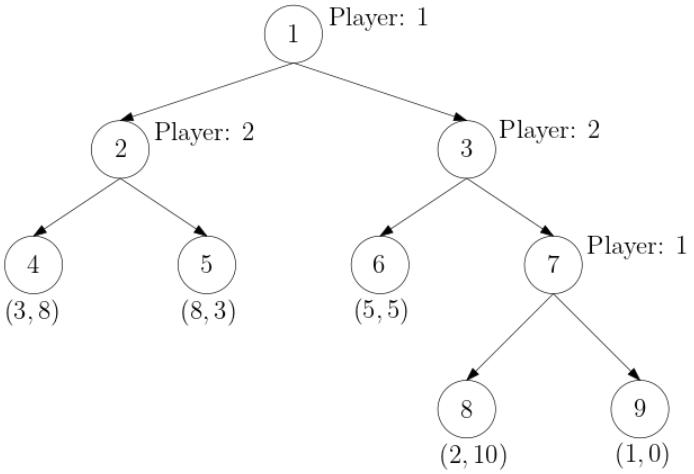

Figure 1: Extensive Game for Problem 3(a)