Problem 3: Vehicle Intersection

In this exercise you have to generate a knowledge base to find the right ordering for up to four vehicles that want to cross an intersection.

Problem Description

For this problem, an intersection can be described as the meeting point of four lanes as can be seen in Figures 1a, 1b, 1c, 1d at the end of this document. Each lane may have a traffic sign, and up to four vehicles (visualized as black dots) may be present at the intersection, at most one per lane.

To solve the task with propositional logic, the problem is modeled with the following propositional variable:

- $V_{n,x}$ which means that the vehicle n gets assigned the number x, with $n = 0, .., 3$ and $x = 0, ..., 4$. For example, $V_{0,1}$ means that the vehicle 0 (bottom) drives first, $V_{2,4}$ means that vehicle 2 (top) drives fourth and $V_{3,2}$ means that vehicle 3 (left) drives second.

- A vehicle that does not exist (empty lane) gets assigned a 0. For example, $V_{2,0}$ means that vehicle 2 (top) does not exist.

Your task for this exercise is to solve the intersection problem using propositional logic, by designing a knowledge base that describes all rules and initial information about the problem.

You should test this implementation on all 5 initial configurations that are given for this exercise. After handing in your implementation, it will be tested on another 10 initial configurations that will be hidden to you to check the correctness of your knowledge base. Only if all scenarios are solved correctly will you have completed the exercise.

Intersection Rules The vehicles obey the standard European traffic laws, but only a few of these are relevant for this exercise:

- A vehicle that has a **Right-of-Way** sign ◇ drives before all other vehicles.

- A vehicle that has a **Stop** sign ⬠ drives after all other vehicles.

- For two vehicles that have no signs, the **Right-before-Left** rule has to be obeyed: The vehicle that arrives from the right of the other vehicle passes first, the other has to wait (an example for this rule is shown in Figure 1a).

- Vehicles can only drive one after the other, so that no two vehicles pass at the same time.

For the sake of simplicity, we assume that every vehicle only drives straight, so that there are no turns. Furthermore, all initial configurations that are given are guaranteed to have unique solution. Finally, unlike for most roads in Europe, in this exercise the Right-of-Way sign applies only on the lane it is assigned to, **not** the opposite lane as well (see Figure 1b).

Since this problem is based on general logic rules, it can be solved using propositional logic.

Programming Framework For this programming exercise a *Jupyter Notebook* will be used. The template for the exercise can be downloaded from ARTEMIS[1]. Since you only have to implement one single function, only minor programming skills in Python are necessary to complete this exercise. The following steps are required to correctly set up the environment for the programming exercise:

1. **Installation of Anaconda and Download of the AIMA Python Code:** If you do not already have the *Jupyter Notebook* environment installed on your machine, the installation is the first step you have to perform. We recommend to install *Anaconda*, since this will set up the whole environment for you. Furthermore, the template for the programming exercise is based on the code from the *AIMA python*[2] project. Instructions on how to install *Anaconda* and the *AIMA python* code can be found in the "*Instructions on how to run the Jupyter Notebooks*" file on Moodle[3].

2. **Retrieving the template:** You can pull the template from ARTEMIS using git from the directory of your choice. You can see the exact link on ARTEMIS, but it should look something like this:

```
git clone https://<your_TUM_ID>@bitbucket.ase.in.tum.de/scm/AI2021LOGIC/ai2021logic-<your_TUM_ID>.git
```

where the two occurences of `<your_TUM_ID>` should be replaced by your TUM ID. You may need to enter you TUM credentials (TUM ID and password). This will download the template into the directory `ai2021logic-exercise`. To avoid issues with relative file paths, we recommend to copy all files contained in this directory to the root-directory of the *AIMA python* project that you downloaded in the previous step.

After completing the above steps, you are all set up to start with the exercise. Open the *Jupyter Notebook* **Logic_Exercise.ipynb** and go through the exercise. The notebook will introduce your task: implement the function which generates the knowledge base necessary to solve an intersection problem. To this end, implement the function *generate_knowledge* in *generate_knowledge.py*. The file *generate_knowledge.py* already contains the empty function `generate_knowledge` and is the only file you have to work on and submit. You are allowed to define other functions in *generate_knowledge.py* if it helps solving the task, but you are not allowed to import any extra packages beside the ones already imported in *generate_knowledge.py*. An example on how to add propositional sentences to the knowledge base in such a way that they are compatible with the remaining code of the notebook is shown in the function `generate_knowledge_example`. The notebook provides you five initialized intersections (configurations) you can use to implement and debug your *generate_knowledge* function. If you execute the notebook, the inference algorithm will solve the chosen intersection problem based on the knowledge base you specified. The inferred order of the vehicles will be visualized at the bottom of the notebook, such that you can easily debug your knowledge base function. If your knowledge base is correct, the inferred order will be the correct solution to the initialized intersection.

Inference Algorithms

The template contains two inference algorithms: the *Forward Chaining* algorithm, which is part of the lecture, and the *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm, which is not part of the lecture.

- **Forward Chaining:** The implementation of the *Forward Chaining* algorithm used in the template can only handle propositional sentences with the following structure:

  - $\alpha_1 \wedge \cdots \wedge \alpha_n$
  - $\alpha_1 \wedge \cdots \wedge \alpha_n \Leftrightarrow \beta_1 \wedge \cdots \wedge \beta_m$
  - $\alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta_1 \wedge \cdots \wedge \beta_m,$

  where α_i and β_i can be a propositional variable S but not the negation of a propositional variable $\neg S$.

- **DPLL:** The *DPLL* algorithm is a backtracking algorithm for inference in propositional logic which was not part of the lecture. If you are curious to discover how the algorithm works you can check out e.g. `https://en.wikipedia.org/wiki/DPLL_algorithm`. The algorithm is able to handle knowledge bases with arbitrary structure, and is therefore recommended for this exercise.

---

[1]`https://artemis.ase.in.tum.de/#/courses/85/exercises`
[2]`https://github.com/aimacode/aima-python`
[3]`https://www.moodle.tum.de/pluginfile.php/2589427/mod_resource/content/1/AIMAinstallation.pdf`

Submission

For the submission, you have to upload the **generate_knowledge.py** file containing your implementation of the knowledge base to ARTEMIS. To do so, copy your solution for **generate_knowledge.py** into the previously cloned `ai2021logic-exercise` directory, and execute the following commands from the directory `ai2021logic-exercise`:

```
git add .
git config user.email "<your.TUM@email.de>"
git config user.name "<Your Name>"
git commit -m "<Write a commit message here.>"
git push
```

You may need to enter you TUM credentials (TUM ID and password) for the upload. We will test your submission on several intersections different from the ones that were provided for helping you implement and debug the function generating the knowledge base. If your implemented function computes the knowledge base such that every test-intersection is solved correctly, you successfully completed this programming exercise. Your code has to compute a valid solution for the test intersections within 5 min on our machine. If your code takes longer to compute a solution, you will fail this submission. Don't worry about the computation time too much as usually the algorithm produces a solution within seconds for our specific exercise. Your submission will be evaluated after the deadline, but until then you can update your solution as many times as you like. The last submitted solution will be graded.

Submissions will close on **22th January at 23:59**.

ATTENTION

- Your code should **not** print anything.
- **Do NOT** rename the submitted file or the function name. If you do you will fail!
- **Do NOT** import any additional modules for your solutions. If you do you will fail!
- Like the rest of the programming exercises, this is an individual project and work **must** be your own. We will use a plagiarism detection tool and any copied code will cancel all bonus points from programming exercises for both the copier and the copied person!
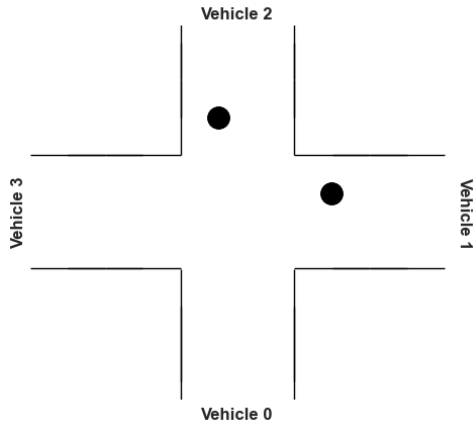
(a) This illustrates the Right-before-Left rule: Since V2 (top) is on the right of V1 (right), it thus drives first.
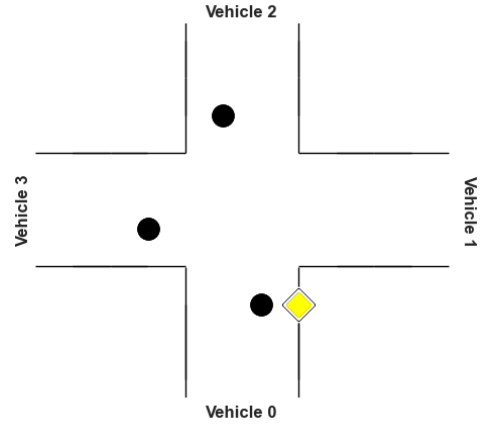
(b) This illustrates the use of the Right-of-Way sign and the Right-before-Left rule: V0 (bottom) drives first since it has a Right-of-Way sign. Since V3 (left) is on the right of V2 (top), V3 drives second, followed by V2 driving third. V1 (right) is empty and thus gets assigned 0.
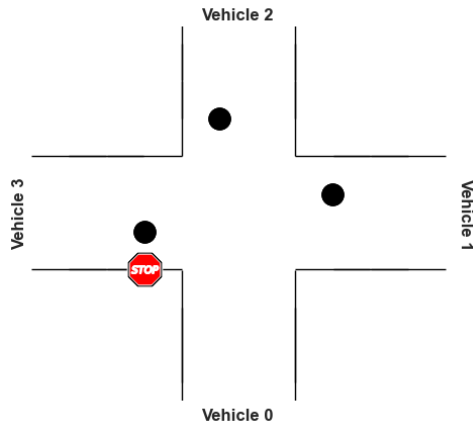
(c) This illustrates the use of the Stop sign: V3 (left) drives last since it has a Stop sign and thus has to wait. Meanwhile, V2 (top) drives first since it is on the right of V1 (right), V1 then drives second, meaning that V3 (left) drives third. V0 (bottom) is empty and thus gets assigned 0.
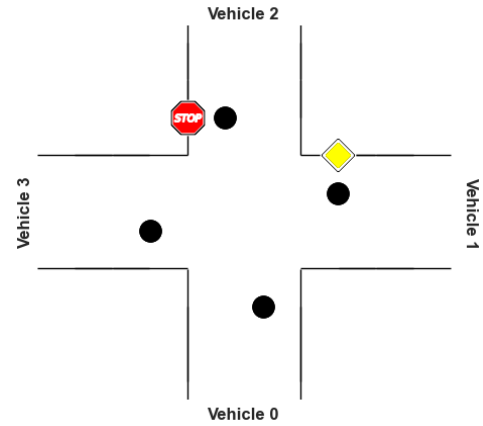
(d) For this intersection, V0 (bottom) drives second, V1 (right) first, V2 (top) fourth, V3 (left) third.

Figure 1: Some initial configurations demonstrating the traffic rules relevant for this exercise.