Grundlagen der Künstlichen Intelligenz

Programming Exercise 3: Propositional Logic

Niklas Kochdumper, Cecilia Curreli and Natalie Stasinski

(last updated 11th December 2019)

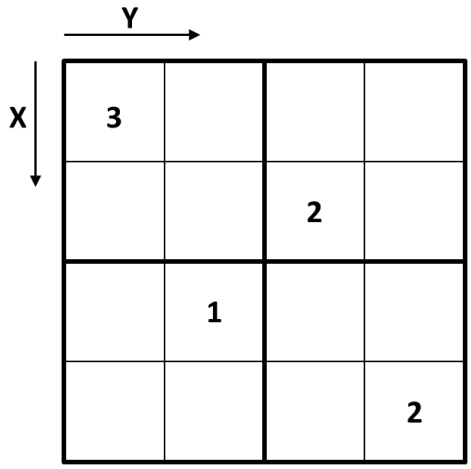
Submission deadline: 17th January 2020, 23:59

Problem 3: Sudoku

This exercise is based on the classic riddle solving game *Sudoku*. Given a square grid of size 4x4 with some initial numbers the goal is to fill in all empty fields with numbers from one to four, with respect to the following rules :

- each subsquare of size 2x2 must contain the numbers 1 to 4
- each row must contain the numbers 1 to 4
- each column must contain the numbers 1 to 4

Since this problem is based on general logical rules it can be solved using propositional logic. If you are not familiar with the Sudoku game you can find lots of explanations of the rules and examples in the internet.



Problem Description

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

- $V_{n,x,y}$ which means that the field [x, y] contains the number n, with $x = 0, .., 3$, $y = 0, ..., 3$, and $n = 1, ..., 4$. For example, $V_{2,1,2}$ means that the field where the second row and the third column meet contains number two

Your task for this programming exercise is to design a knowledge base that describes all rules and initial information about the sudoku game with the propositional variable from above. If the knowledge base is specified correctly, then the given sudoku will be solved correctly.

Programming Framework

For this programming exercise *Jupyter Notebooks* will be used. The template for the exercise can be found in Moodle. 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:** 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. Instructions on how to install *Anaconda* can be found in the "*AIMAcode Installation Instructions*" file in Moodle.
2. **Download of the AIMA python code:** The template for the programming exercise is based on the code from the *AIMAcode* project. Therefore, you first have to download the code from this project before the template can be used. Instructions on how to obtain the code from this project can be found in the "*AIMAcode Installation Instructions*" file in Moodle.
3. **Download of the template:** Download the .zip-folder with the template from Moodle. To avoid issues with the relative file paths, we recommend to copy all files contained in the .zip-folder into the root-directory of the *AIMAcode* 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* **Exercise.ipynb** and go through the exercise. The notebook will introduce your task: implement the function which generates the knowledge base necessary to solve a sudoku map given a initialized sudoku. 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 sudoku (sudoku maps) you can use to implement and debug your *generate_knowledge* function. If you execute the notebook the inference algorithm will solve the chosen sudoku map based on the knowledge base you specified. The inferred sudoku 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 sudoku will be the correct solution to the initialized sudoku.

Inference Algorithms

The templates contains the two inference algorithms *Forward Chaining*, which is part of the lecture, and the *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm, which is not part of the lecture. We recommend to choose from these two algorithms the one that is better suited for the structure of the knowledge base you defined since this could lead to a significant speed-up for inference:

- **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 are either a propositional variable S or the negation of a propositional variable $\neg S$.
- **DPLL:** The *DPLL* algorithm is another 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.

Submission

For the submission, you have to upload the **generate_knowledge.py** file containing your implementation of the knowledge base in Moodle. We will test your submission on several sudoku maps 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-sudoku is solved correctly, you successfully completed this programming exercise. Your code has to compute a valid solution for the test sudoku maps within 5 min on our machine. If your code takes longer to compute a solution, your 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.