Principles of AI Planning

R. Mattmüller, Prof. Dr. B. Nebel
T. Keller
Winter Term 2011/2012

University of Freiburg
Department of Computer Science

Project 1

Due: November 29th, 2011

In the project exercises, we will both use and alter Pyperplan, which you have already encountered on the first exercise sheet (and which can be downloaded at `https://bitbucket.org/malte/pyperplan` in case you haven't). As Pyperplan is written in Python 3, all exercises can only be implemented in Python 3. All solutions should be sent by email to tkeller@informatik.uni-freiburg.de and will be tested to verify their functionality.

Exercise 1.1 (Manual compilation to STRIPS, 4 points)

Download the `aip-teaching` domain from the website of the course. The objective of `aip-teaching` planning problems is for `students` to understand the principles of important AI planning `topics` in order to pass an exam at the end of the course. To do so, students must achieve different levels of understanding both practically and theoretically. Additionally, if a student missed too many of the lectures, he/she will need to learn extra hard to pass the exam.
Starting Pyperplan on this domain will lead to several errors. These are caused by the actions `learn-theoretical-level3` and `learn-practical-level3`, which have preconditions that contain a negated predicate, and by the precondition of `solve-exercise` which contains a disjunction. As this is not allowed in STRIPS it isn't supported by Pyperplan.
Modify the domain and problem files such that they are supported by Pyperplan and functionally equivalent, i.e. create the domain `aip-teaching-strips.pddl` where neither disjunctions nor negated predicates occur in preconditions, and adapt the initial state in a new problem file `aip-teaching-problem-strips.pddl` accordingly.
For negated predicates in preconditions you can use the procedure that was presented in the lecture to convert operators to *positive normal form.* To remove the disjunction you will have to split the operator `solve-exercise` in two operators `solve-exercise_1` and `solve-exercise_2`, one applicable in the states where theoretical understanding is on the highest level and practical on a rather low one, and the other applicable when practical understanding is on the highest level and theoretical on a medium one. Note that the renaming of the actions is necessary as Pyperplan requests unique names for all actions.
The solution to this exercise are the modified domain and problem files `aip-teaching-strips.pddl` and `aip-teaching-problem-strips.pddl`.

Exercise 1.2 (Compilation to STRIPS in Pyperplan, 8 points)

Download the `compilation-pyperplan.tar.gz` archive from the website and unpack it to the root folder of you pyperplan directory, thereby overwriting the files `src/pyperplan.py`, `src/pddl/pddl.py` and `src/pddl/tree_visitor.py`. We prepared the following changes to the code:

- We introduced a class `NegPredicate`, which is used *only* to create negated predicates in preconditions.
- In the original version, a precondition in Pyperplan must always be a conjunction, which is represented as a list of objects of type `Predicate`. We adapted the parser to expect preconditions in disjunctive normal form (DNF) (it does *not* support arbitrary disjunctions in preconditions!), so a precondition is now represented as a list of lists of `Predicates` (and now also `NegPredicates`), where each entry in the list of lists corresponds to a conjunction and the whole struct represents a formula in DNF.

- We added a function `_compile_to_strips(problem)`, which calls `_remove_disjunctions(problem)` and `_remove_negations(problem)`. Both functions take an instance of type `Problem` as a parameter, which is defined in `pddl/pddl.py` and contains everything that must be accessed for this exercise (including an instance of class `Domain` that also contains some members that must be accessed).
- The function `_adapt_initial_state(task)` alters the initial state by adding all necessary atoms that are created to get rid of negated predicates in preconditions.

The functions `_remove_disjunctions(problem)` and `_remove_negations(problem)` are incomplete. To fill the gaps, you need to do the following:

- In `_remove_disjunctions(problem)`, actions with preconditions that are lists of `Predicates` and `NegPredicates` have to be created.
- In `_remove_negations(problem)`, all preconditions and effects have to be adapted according to the algorithm to convert operators to *positive normal form* that was presented in the lecture.

Please hand in the modified file `pyperplan.py` (and all other files you might have modified) as solution to this exercise.

Exercise 1.3 (Goal count heuristic in Pyperplan, 8 points)

After working with the frontend of Pyperplan, we will now see how to implement our own heuristic function in Pyperplan, and how to compare its efficiency to that of other heuristics.
The goal count heuristic is a very simple heuristic already used in the *Stanford Research Institute Problem Solver* (Fikes and Nilsson) in 1971. It requires the set of goal states to be defined by a conjunction of literals. The goal count heuristic estimates the distance from a given state to the goal by counting *how many literals from the goal formula are unsatisfied in that state.* E.g., if the goal formula is $\gamma = a \land \neg c \land e$ and the current state s satisfies the literals a, b, c, $\neg d$ and $\neg e$, then $h_{GC}(s) = 2$ (as c and e do not have the desired value).
The file `goal_count.py` was added to the folder `src/heuristics/` in your local copy of Pyperplan when you extracted the archive for Exercise 1.2. Fill in the missing `__call__` method that is supposed to return the heuristic value given a search node `node` containing a state `node.state`. In Pyperplan, a state is simply a set of positive literals (those that are true in that state). Note that unlike in the example above, here we do not have any negated literals in the goal. This further simplifies the implementation of the goal count heuristic.
We added the goal count heuristic to the dictionary of heuristics defined at the beginning of `src/pyperplan.py` under the name `'hgc'`. Compare your implementation of the goal count heuristic to the landmark cut heuristic (`lmcut`), the FF heuristic (`hff`), the max heuristic (`hmax`) and the additive heuristic (`hadd`) on the `aip-teaching` domain. Write down the plan lengths and the numbers of node expansions using greedy best first search for those heuristics that find a plan after 2 minutes.
Please hand in the modified file `goal_count.py` and the results of your experiments as solution to this exercise.

Note: The exercise sheets may and should be worked on in groups of two students. Please state both names on your solution (this also holds for submissions by e-mail).