FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE - CS161
Spring 2022

Assignment 6 – Due May 11 at 11:55pm

Please submit your homework via **Bruinlearn**. The Lisp file should be named **hw6.lsp**. The "written" part of the assignment should be in a file named **hw6.txt**. Use the following convention for logic symbols: | (for disjunction), & (for conjunction), ~ (for negation), => (for implication), <=> (for equivalence), E (for existential quantification, e.g., (E x,y) (loves(x, y))), and A (for universal quantification, e.g., (A x, y) (loves(x, y))). You may assume the normal order of operations as described in your text, but you may use parentheses to override this order or to make things more clear.

Note: You might find it useful to start looking at Problem 3 first, as it only requires knowledge about Propositional Logic.

1. (10 pts) For each pair of atomic sentences, give the most general unifier if it exists:

(a) P(A, A, B), P(x, y, z).
(b) Q(y, G(A, B)), Q(G(x, x), y).
(c) R(x,A,z) , R(B,y,z).
(d) Older (Father(y), y), Older (Father(x), John).
(e) Knows(Father(y),y), Knows(x,x).

2. (30 pts) Consider the following sentences:

- John likes all kinds of food.
- Apples are food.
- Chicken is food.
- Anything someone eats and isn't killed by is food.
- If you are killed by something, you are not alive.
- Bill eats peanuts and is still alive. *
- Sue eats everything Bill eats.

(a) Translate these sentences into formulas in first-order logic.
(b) Convert the formulas of part (a) into CNF (also called clausal form).
(c) Prove that John likes peanuts using resolution.
(d) Use resolution to answer the question, "What does Sue eat?"
(e) Use resolution to answer (d) if, instead of the axiom marked with an asterisk above, we had:

- If you don't eat, you die.
- If you die, you are not alive.
- Bill is alive.

3. (60 pts) In this problem, you will write a Lisp program that converts graph coloring problems into SAT problems and use a SAT solver to solve them. We have broken the task into multiple functions and provided you with some basic code for file I/O and for gluing all the functions together (see **hw6-skeleton.lsp**).
For each graph coloring problem, each node will be represented by a positive integer (node index). Each color is also represented by a positive integer (color index).

Similarly, for a SAT problem, each variable is represented by a positive integer (variable index). As a result, a positive integer is used for a positive literal and a negative integer is used for a negative literal. A clause is simply a list containing the literals of the clause. A CNF formula is then simply a list of clauses.

For example, if variable a has index 1, variable b has 2, and variable c has index 3, then the clause (a ∨ ¬ b ∨ ¬c) is represented as the list (1 -2 -3). The CNF formula (a ∨ b ∨ c) ∧ (¬ a ∨ b ∨ ¬c) is represented as ((1 2 3) (-1 2 -3)). Of course, the order of clauses and literals in each clause does not matter.

Here are your tasks:

i) Write a function called node2var that takes a node index (n), a color index (c), and the maximum color index (k). This function should return the index of the propositional variable that represents the constraint: “node n receives color c” (with k colors being considered). Use the following conversion convention:

$$\textit{variable index} = (n - 1) * k + c$$

ii) Write a function called at-least-one-color that takes a node index (n), a color index (c), and the maximum color index (k). This function should return a clause that represents the constraint: “node n must be colored with at least one color whose index comes from the set {c, c+1, …, k}.”

iii) Write a function called at-most-one-color that takes the same arguments as at least-one-color. This function should return a list of clauses that represent the constraint: “node n must be colored with at most one color whose index comes from the set {c, c+1, …, k}.” **You may write your own helper functions.**

iv) Write a function called generate-node-clauses that takes a node index (n) and the maximum color index. This function should return a list of clauses that constrain node n to be colored with exactly one color whose index is in the set {1, 2, …, k}.

v) Write a function called generate-edge-clauses that takes an edge (x y) and the maximum color index (k). An (undirected) edge is simply a list of two node indices. This function should return a list of clauses that prohibit nodes x and y from having the same color in the set {1, 2, …, k}. **You may write your own helper functions.**

After finishing all the above parts, you should be able to convert a graph coloring problem into a SAT problem. To do so, call

```
(graph-coloring-to-sat <graph filename> <SAT filename> <max color index>)
```

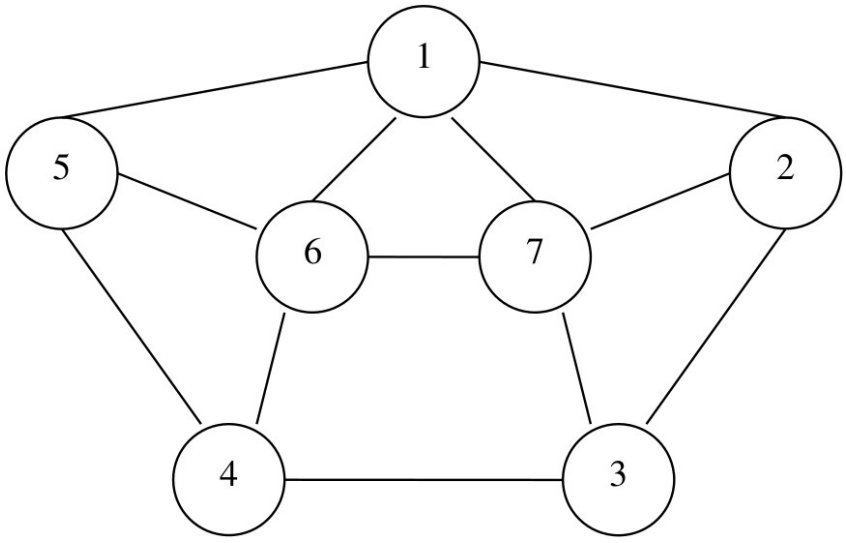
The graph filename is the name of the input graph file. The SAT filename is the name of the output file you want the program to write to. Max color index is simply the number of colors being considered in the problem.

The graph file has the following format:

- The first line contains 2 numbers. The first one is the number of vertices in the graph, the second one is the number of edges.
- Each subsequent line describes an edge. An edge is represented by two numbers—the node indices of the two nodes linked by the edge.

Now that you have a converting program working, you will use it to convert some actual graph coloring problems into SAT problems and solve them with a SAT solver.

First, consider the following graph (whose nodes are labeled with their node indices):



A graph file for this graph is also provided (graph1.txt). Convert the graph coloring problem of this graph with 3 colors into a SAT instance using the program you wrote.

Then, download the RSat SAT solver from (http://reasoning.cs.ucla.edu/rsat/). Read the manual carefully. Use RSat to solve the SAT instance obtained above. **(1). Is the instance satisfiable?**

Do the conversion again, this time, with 4 colors. Use RSat to solve this new SAT instance. **(2). Is the instance satisfiable?**

(3). What do the answers of these two SAT instances tell you about the graph coloring problem of the above graph? Can you give a solution (a coloring) to the graph coloring problem of the above graph based on the results of Rsat?

Now, use a similar approach to solve the graph coloring of the graph described in graph2.txt. **(4). What is the minimum number of colors required to properly color this graph?**

Submit your lisp file (named hw6.lsp) via Bruinlearn. Put all non-programming answers (including those to questions 3-(1) to 3-(4) shown in bold) in hw6.txt and submit both files through Bruinlearn.