

Problem Set 1

Handed Out: 2/3/2009

Due: 2/17/2009

1. [Sentence Translation]

(a) Translate the following sentences into first-order logic. Define your own predicates, functions, constants, etc. Use symbols that make sense. You can also use the operator $=$.

- i. There is a barber who shaves all men in town who do not shave themselves.
- ii. Only one student failed this class last year.
- iii. I won't be able to finish the homework unless someone helps me with the logic problems.

(b) There is often confusion because the \Rightarrow connective does not correspond directly to the English *if... then ...* construction. The following English sentences use *and*, *or*, and *if* in ways that are quite different from first-order logic. For each sentence, give both a translation into first-order logic that preserves the intended meaning in English, and a straightforward translation (as if the logical connectives had their regular first-order logic meaning). Specify whether the straightforward translation is valid, satisfiable or invalid.

- i. Maybe I'll go to the movie and maybe I won't.
- ii. Either I get an A in this class or my GPA will drop below 3.5.
- iii. The Academy Award is on TV tonight, if you're interested.
- iv. If you lived here you would be home now. If you were home now, you would not be here. Therefore, if you lived here you would not be here.

2. [Resolution Refutation]

For the following sentences, first translate them into first order logic, then convert them into conjunctive normal form.

- (a) Marcus was a man.
- (b) Marcus was Pompeian.
- (c) Marcus was born in 40 A.D.
- (d) All men are mortal.
- (e) All Pompeians died when the volcano erupted in 79 A.D.
- (f) No mortal lives longer than 150 years.
- (g) It is now 2004.
- (h) Alive means not dead.
- (i) If someone dies, then he is dead at all later times.

Now use resolution refutation to prove the following statement.

Marcus is not alive in 2004.

3. [Mine Sweeper and Logic]

This is a machine problem. In this problem, you are to write a mine sweeper solver. Well, not really, because the original problem has been simplified, and only a certain set of initial configurations of the board could be solved by your program. Let us call our game one-mine sweeper. Your program will use the concepts and techniques you learned in class on logic deduction.

• Rules for The Game

The rules for one-mine sweeper are as follows:

- There is a $m \times n$ grid board, and each of the mn squares may or may not have a mine in it. The total number of mines is known to the player.
- There is either no mine or only 1 mine in the surrounding squares of any square on the board. (Note: this is a restriction that the original game does not impose.)
- In the beginning of the game, all squares are covered. The player is given a single square that is guaranteed to be safe (*i.e.*, no mine in it.)
- The player can choose to uncover a square. If the square contains a mine, the player dies and the game ends; if the square does not contain a mine, the player is informed whether there is a mine in the surrounding 8 squares of this uncovered square.
- The player wins when she uncovers all the safe squares and leaves only the squares that contain mines covered.

• Your Task

Your program will simulate both the computer and the human player. On one hand, the player part of your program will make deduction based on the current knowledge about the board, and uncover safe squares or flag squares that have mines. On the other hand, the computer part of your program will provide certain information when a safe square is uncovered, or end the game when a unsafe square is uncovered due to the wrong judgment of the player. Now let us break down the task into parts and explain them one by one.

• Represent the Board

Given a file that tells where the mines are, your first job is to write first-order logic sentences to represent the board. The file tells you the size of the board, the number of mines on the board, and the x and y coordinates of the squares that contain those mines.

Your program should read in this file, and translate the information into FOL sentences with the following predicates:

- $Mine([x, y])$: square $[x, y]$ has a mine in it.
- $Adjacent0([x, y])$: square $[x, y]$ has no adjacent mine.
- $Adjacent1([x, y])$: square $[x, y]$ has 1 adjacent mine in its surrounding squares.
- $N([x, y], [u, v])$: square $[x, y]$ is adjacent to square $[u, v]$ and is on the north of $[u, v]$.
- $E([x, y], [u, v])$: square $[x, y]$ is adjacent to square $[u, v]$ and is on the east of $[u, v]$.
- $NE([x, y], [u, v])$: square $[x, y]$ is adjacent to square $[u, v]$ and is on the northeast of $[u, v]$.

- *SE* ($[x, y], [u, v]$): square $[x, y]$ is adjacent to square $[u, v]$ and is on the southeast of $[u, v]$.

Keep two knowledge bases to store these facts. Knowledge base I stores all *Valid*, *Invalid*, *N*, *E*, *S*, *W*, *NE*, *SE*, *SW*, and *NW* sentences, while knowledge base II stores all *Bomb*, *Adjacent0*, and *Adjacent1* sentences. Later sentences in knowledge base II will gradually be added to knowledge base I.

• Write Rules for Inference

Your next job is to write the rules to do inference on knowledge base I. In addition to the predicates defined above, add the following predicate:

– *Safe*($[x, y]$): square ($[x, y]$) is safe and therefore can be uncovered.

Your rules should all be Horn sentences so that later you could use forward chaining to do the inference. Since all you care about is which squares are safe and while contains mins, all your rules should be of the following format:

$$P_1 \wedge P_2 \wedge \dots \wedge P_k \Rightarrow \text{Safe}([x, y])$$

or

$$P_1 \wedge P_2 \wedge \dots \wedge P_k \Rightarrow \text{Bomb}([x, y])$$

where each P_i is some predicate with variables.

• Build an Inference Engine

Now that you have facts and rules in your knowledge base I, you can write an inference engine to expand the facts and find out where the mines are. Your inference engine should make no assumptions about the predicate names, number of variables, etc. It can assume, however, that each rule is of Horn form. Your inference engine needs some substitution mechanism to replace the variables in the rules with the constants in the facts. Your inference engine should use forward chaining algorithm to expand knowledge base I.

• Link Everything Together

Feed the facts and the rules in knowledge base I to your inference engine. We will give you a square $[x_0, y_0]$ that is guaranteed to be safe. Start with finding *AdjacentX* ($[x_0, y_0]$) in knowledge base II and adding this piece of information to knowledge base I.

Now expand your knowledge base one. Whenever your inference engine decides that some square $[x, y]$ is safe, and *AdjacentX* ($[x, y]$) is not already in knowledge base I, your program should look for some *AdjacentX* ($[x, y]$) in knowledge base II, and add it to knowledge base one. (If, however, no *AdjacentX* ($[x, y]$) is in knowledge base II, but *Bomb*($[x, y]$) is, then either you have written an incorrect rule, or your inference engine has made a mistake.) (*Hint* : Depending on your rules, it may not be necessary to keep the sentence *Safe*($[x, y]$) in your knowledge base I because you have replaced it with *AdjacentX* ($[x, y]$).)

Continue the process until no new facts could be added to knowledge base I.

• **What to Turn In**

(a) A hardcopy of your code.

b) We would like to see the rules you used for inference, so turn in a copy of all the rules you write as described in section *Write Rules for Inference*. Use the exact formats as described in that section.

(c) We will provide you with an initial configuration of the board. You need to turn in a description of which mines your program has found. Simply list the coordinates of those squares, in ascending order of x coordinate (and ascending order of y coordinate when there is a tie.) For example: [2, 4], [3, 2], [3, 8], . . .