CS 7750: Report for bonus assignment 3

Chanmann Lim
Sam Jonesi

November 13, 2014

I. Implementation description

The project is implemented in Java programming language and make use of the existing code available online from the companion website of the book "Artificial Intelligent - A Modern Approach 3rd Edition" `https://code.google.com/p/aima-java/`.

The code infrastructure for dealing with propositional logic representation and reasoning has already been developed. We are here taking the advantages of the codebase to implement our `WampusWorld Inference Program` by feeding knowledges into the knowledge base then using the existing Truth table entailments algorithm to make inferences for Figure. 7.3(a), 7.3(b), and 7.4(a) from the textbook. In general, there are three possibilities in solving propositional inference program 1) Truth table enumeration entailment 2) Forward or backward chaining and 3) Propositional logic resolution. However, the 2 latter solutions require certain forms of propositional logic sentence representations known as Horn form for proving by Forward or backward chaining and Conjunctive Normal Form (CNF) for Propositional logic resolution. In addition, the scope of the assignment to make inferences about the 4×4 wumpus world lead to discrete and small knowledge base of the program and Truth table enumeration seem to yield better performance than other alternatives and can be considered as a perfect fit for the given task.

The single most important class to our implementation is the `KnowledgeBase` class and after instantiation of the class we can add the knowledges about our WumpusWorld into the knowledge base using `tell(aSentence)` method which accepts a `String` argument as a sentence of propositional logic and behind the scene the `KnowledgeBase` class has propositional logic parsing mechanism accounting for converting the given string to `Sentence` type which is necessary for various inferencing algorithms we might use later then we use `askWithTTEntails(queryString)` method to make inferences.

Despite its simplicity, ones need to understand the convention used to encode the logical connective long with symbols for using in the `tell(aSentence)` method introduced above. From the documentation in the `Connective.java` class we can see that the five common connectives are represented by:

```
1. ~   (not).
2. &   (and).
3. |   (or).
4. =>  (implication).
5. <=> (biconditional).
```

in this regard, we can represent there is no Breeze in square [1, 1] by `~B11`, and Breeze in square [2,1] biconditional there is a pit in square [1, 1] or [3, 1] or [2, 2] by `B21 <=> P11 | P31 | P22`.

Figure. 7.3(a) shows that there is no Breeze in square [1, 1] so we can feed in the knowledge by `tell("~B11")`, but we also need to provide what's breezy in square [1, 1] mean (there is at least a pit in the adjacent squares) so we can feed in `tell("B11 <=> P12 | P21")` and also feed in the definitions of OK in square [1, 2] by `tell("OK12 <=> ~P12")` and OK in square [2, 1] by `tell("OK21 <=> ~P21")` before we could be able to make inference about them and finally we use `askWithTTEntails("OK12")` for check if it is safe in square [1, 2] and do the same for square [2, 1]. Using this information, the program found that no breeze in [1, 1] implied that there was no pit in [1, 2] or [2, 1]. It then implied that if there was no pit in [1, 2] and [2, 1] that those squares are safe. To simplify the problem we haven't consider Wumpus and Stench percept for this task.

Figure. 7.3(b) we told the knowledge base that a breeze in 2,1 biconditionally implies a pit in 11, 31, or 22. We then told the knowledge base that if 11, 12, and 21 are safe, then there is no pit in them. Finally, we told the knowledge base that there was a breeze in 21, and that 11 is safe. From there, it determined that a pit in 3,1 or 2,2 could be possible, but is not necessarily true.

Figure. 7.4(a) we told the knowledge base that a stench in 12 biconditionally implied that there is a wumpus in 11, 22, or 13. We told the knowledge base that is there is a breeze in 21, there is a pit in 11, 31, or 22. We then told the knowledge base that an ok in 11, 12, 21, or 22 biconditionally implies that

there is no wumpus and no pit in the respective squares. Finally, we told the knowledge base that 12 had a stench, 11, 12, 21, 22 were ok, and that there was a breeze in 21. Finally, we asked the knowledge base if a wumpus in 13, ok in 22, or a pit in 31 were entailed. It then found that there was a wumpus in 13 because there was a stench in 12, but 22 was safe. It found that 22 was safe because we told it so. Finally, it a pit in 31 entailed the knowledge base, because there was a breeze in 21, but 11 and 22 were ok.

II. Code:

../src/main/java/bonus3/WumpusWorld.java

```
package bonus3;

import aima.core.logic.propositional.kb.KnowledgeBase;

public class WumpusWorld {

    public static void main(String[] args) {
        WumpusWorld ww = new WumpusWorld();

        ww.print_7_3_a();
        ww.print_7_3_b();
        ww.print_7_4_a();
    }

    public void print_7_3_a() {
        KnowledgeBase kb = new KnowledgeBase();

        kb.tell("B11 <=> P12 | P21");
        kb.tell("OK11 <=> ~P11");
        kb.tell("OK12 <=> ~P12");
        kb.tell("OK21 <=> ~P21");

        kb.tell("OK11");
        kb.tell("~B11");

        printEntailments(kb, "OK12", "OK21");
    }

    public void print_7_3_b() {
        KnowledgeBase kb = new KnowledgeBase();

        kb.tell("B21 <=> P11 | P31 | P22");
        kb.tell("OK11 <=> ~P11");
        kb.tell("OK12 <=> ~P12");
        kb.tell("OK21 <=> ~P21");

        kb.tell("B21");
        kb.tell("OK11");

        printEntailments(kb, "P31", "P22");
    }

    public void print_7_4_a() {
        KnowledgeBase kb = new KnowledgeBase();

        kb.tell("S12 <=> W11 | W22 | W13");
        kb.tell("B21 <=> P11 | P31 | P22");
        kb.tell("OK11 <=> ~W11 & ~P11");
        kb.tell("OK12 <=> ~W12 & ~P12");
        kb.tell("OK21 <=> ~W21 & ~P21");
        kb.tell("OK22 <=> ~W22 & ~P22");

        kb.tell("S12");
        kb.tell("B21");
        kb.tell("OK11");
        kb.tell("OK12");
        kb.tell("OK21");
        kb.tell("OK22");

        printEntailments(kb, "W13", "OK22", "P31");
```

```
    }

    private void printEntailments(KnowledgeBase kb, String... queries) {
        StringBuilder sb = new StringBuilder();

        sb.append("KB: ").append(kb.toString()).append("\n");
        for (String queryString : queries) {
            boolean valid = kb.askWithTTEntails(queryString);
            boolean invalid = kb.askWithTTEntails("~" + queryString);

            sb.append("  -> ")
              .append(queryString)
              .append(" = ")
              .append(valid == invalid ? "?" : valid ? "Yes" : "No");
            sb.append("\n");
        }
        sb.append("\n");

        System.out.println(sb.toString());
    }
}
```

III. Program output:

```
KB: (B11 <=> P12 | P21) & (OK11 <=> ~P11) & (OK12 <=> ~P12) & (OK21 <=> ~P21) &
OK11 & ~B11
  -> OK12 = Yes
  -> OK21 = Yes


KB: (B21 <=> P11 | P31 | P22) & (OK11 <=> ~P11) & (OK12 <=> ~P12) & (OK21 <=> ~P21) &
B21 & OK11
  -> P31 = ?
  -> P22 = ?


KB: (S12 <=> W11 | W22 | W13) & (B21 <=> P11 | P31 | P22) & (OK11 <=> ~W11 & ~P11) &
(OK12 <=> ~W12 & ~P12) & (OK21 <=> ~W21 & ~P21) & (OK22 <=> ~W22 & ~P22) &
S12 & B21 & OK11 & OK12 & OK21 & OK22
  -> W13 = Yes
  -> OK22 = Yes
  -> P31 = Yes
```