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" at `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 a 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 and no Stench in square [1, 1] so we can feed in the knowledge by `tell("~B11")` and `tell("~S11")`, but we also need to provide what's breezy and stinky in square [1, 1] mean (there is at least a pit or wumpus in the adjacent squares) so we can `tell("B11 <=> P12 | P21")` and `tell("S11 <=> W12 | W21")` also feed in the definitions of OK in square [1, 2], and [2, 1] by `tell("OK12 <=> ~W12 & ~P12")` and `tell("OK12 <=> ~W21 & ~P21")` respectively before we could make any inferences by `askWithTTEntails("OK12")` and `askWithTTEntails("OK21")` to see if it is indeed safe (OK) in those squares. 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 and the same logic applied to imply there no wumpus in those squares finally the program concluded that those squares are safe.

Figure. 7.3(b) we told the knowledge base that a breeze in [2, 1] biconditionally implies a pit in [1, 1], [3, 1], or [2, 2] by `tell("B21 <=> P11 | P31 | P22")`. We then told the knowledge base that there exist a breeze in [2, 1] by `tell("B21")`. 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 [1, 2] biconditionally implies that there is a wumpus in [1, 1], [2, 2], or [1, 3]. We told the knowledge base that a breeze in [1, 2] if and only

if there is a pit in [1, 1], [2, 2], or [1, 3]. We then told the knowledge base that a stench or a breeze in [2, 1] biconditionally implies that there is a wumpus or a pit respectively in the adjacent squares to [2, 1] then we told the knowledge base that [1, 2] has a stench but not a breeze, and that there was a breeze but not a stench in [2, 1]. Finally, we asked the knowledge base if a wumpus in [1, 3], ok in [2, 2], and a pit in [3, 1] were entailed. It then found that there was a wumpus in [1, 3] because there was a stench in [1, 2], but [1, 1], [2, 2] were safe. It found that [2, 2] was safe because there is no stench in [2, 1] and no breeze in [1, 2] imply no wumpus and no pit in [1, 1] and [2, 2]. Finally, it a pit in [3, 1] entailed the knowledge base, because there was a breeze in [2, 1], but [1, 1] and [2, 2] were ok.

Helper function: `printEntailments(kb, queries)` is a helper function that print the output of the entailments provided a knowledge base and a list of queries. The function iterates the queries and asks entailment for each query string with `askWithTTEntails(queryString)` and its negation `askWithTTEntails("~" + queryString)`; if they both return false implied that the knowledge base could not make clear and solid inference and output "?" otherwise the result `askWithTTEntails(queryString)` and its corresponding queryString will be appended to the output string which was constructed using `StringBuilder` class.

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("S11 <=> W12 | W21");
        kb.tell("OK12 <=> ~W12 & ~P12");
        kb.tell("OK21 <=> ~W21 & ~P21");

        kb.tell("~B11");
        kb.tell("~S11");

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

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

        kb.tell("B21 <=> P11 | P31 | P22");
        kb.tell("B21");

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

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

        kb.tell("S12 <=> W11 | W22 | W13");
        kb.tell("B12 <=> P11 | P22 | P13");
        kb.tell("B21 <=> P11 | P31 | P22");
        kb.tell("S21 <=> W11 | W31 | W22");
        kb.tell("OK11 <=> ~W11 & ~P11");
        kb.tell("OK22 <=> ~W22 & ~P22");
```

```
        kb.tell("S12");
        kb.tell("~B12");
        kb.tell("~S21");
        kb.tell("B21");

        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) & (S11 <=> W12 | W21) & (OK12 <=> ~W12 & ~P12) &
(OK21 <=> ~W21 & ~P21) & ~B11 & ~S11
  -> OK12 = Yes
  -> OK21 = Yes


KB: (B21 <=> P11 | P31 | P22) & B21
  -> P31 = ?
  -> P22 = ?


KB: (S12 <=> W11 | W22 | W13) & (B12 <=> P11 | P22 | P13) & (B21 <=> P11 | P31 | P22) &
(S21 <=> W11 | W31 | W22) & (OK11 <=> ~W11 & ~P11) & (OK22 <=> ~W22 & ~P22) &
S12 & ~B12 & ~S21 & B21
  -> W13 = Yes
  -> OK22 = Yes
  -> P31 = Yes
```