Informatics 2D. Coursework 1: Propositional Model Checking and Satisfiability

Kobby Nuamah, Michael Rovastos

January 28, 2016

1 Introduction

The objective of this assignment is to help you understand inference procedures using Propositional Logic. You will implement, use and evaluate the inference problems and their algorithms using *Haskell*.

You should download the following file from the Inf2D coursework page:

http://www.inf.ed.ac.uk/teaching/courses/inf2d/coursework/Inf2dAssignment1.tar.gz

Use the command **tar xvf Inf2dAssignment1.tar.gz** to extract the files contained within. This will create a directory named *Inf2dAssignment1/* containing *Inf2d.hs* and *Main.hs* files for the assignment.

You will submit a version of the *Inf2d.hs* file containing your implemented functions and a small report. Remember to add your **matriculation number** at the top of the file.

The deadline for the assignment is:
Thursday, 3rd of March 2016 at **4 pm**.

Submission details can be found in Section 12

If your Haskell is quite rusty, you should revise the following topics:

- Recursion
- Currying
- Higher-order functions
- List Processing functions such as map, filter, foldl, sortBy, etc
- The Maybe monad

The following webpage has lots of useful information if you need to revise Haskell. There are also links to descriptions of the Haskell libraries that you might find useful:

Haskell resources: http://www.haskell.org/haskellwiki/Haskell

In particular, to read this assignment sheet, you should recall the Haskell type-definition syntax. For example, a function **foo** which takes an argument of type **Int** and an argument of type **String**, and returns an argument of type **Int**, has the type declaration **foo :: Int** $\rightarrow$ **String** $\rightarrow$ **Int**. Most of the functions you will write have their corresponding type-definitions already given.

2 Important Information

There are the following sources of help:

- Attend lab sessions
- Read "Artificial Intelligence: A Modern Approach" Third Edition, Russell & Norvig, Prentice Hall, 2010 **(R&N)** Chapter 7 Logical Agents or

  "Artificial Intelligence: A Modern Approach" Third Edition, Pearson New International Edition, Russell R & Norvig P, Pearson, 2014 **(NIE)** Chapter 6 Logical Agents
- Email Kobby Nuamah (k.nuamah@ed.ac.uk)

- Read all emails sent to the Inf2D mailing list regarding the assignment.

Also ...

- Comment your Haskell code.
- Make sure your code compiles successfully before submitting.
- Your code will be tested using the libraries and supporting files provided in the assignment folder.
- Dont change the names or type definitions of the functions you are asked to implement.

3 Getting Started

In this assignment, you will be implementing some inference procedures for propositional logic. Specifically, you will be implementing the Truth-Table Enumeration algorithm for deciding propositional entailment, as well as the DPLL (Davis-Putnam-Logemann-Loveland) algorithm for checking satisfiability of a sentence in propositional logic. This assignment will guide you through the implementation by providing you function signatures which you must complete for your inference programs to work.

You can also test your implementation by using the support code provided in the assignment folder. You can load the main file of the Haskell program using the commands:

```
:cd <path to your project directory>
:load "Main.hs"
```

This will compile all dependent modules of the program, including the file in which you will be implementing the algorithms, ***Inf2d.hs*** You can also launch the menu of the test program using:

```
main
```

at the command prompt of your Haskell GHCi.

Note: You should change the variable

studentId::String

in Inf2d.hs from *undefined* to your student matriculation number, else you will get an exception when you run the test program.

4 Representation

All logical sentences in this coursework will be represented in *Conjunctive Normal Form (CNF)*. That is, every sentence is expressed as a conjunction of clauses, where causes are disjunctions of literals. You should refer to R& N Chapter 7 Section 5.2 or NIE Chapter 6 (Logical Agents) Section 5.2.

In Haskell, we will represent symbols as *Strings*. So the symbol P is represented as "P" and the negation symbol $\neg P$ is represented as "-P".

Clauses (disjunction of literals/symbols) will be represented as a list of symbols. For example, the clause $P \vee Q$ is represented as ["P","Q"], and $A \vee \neg B \vee C$ is represented as ["A","-B","C"].

Conjunctions are represented as lists of clauses. So we express $A \wedge B$ as as [["A"],["B"]].

Putting all these together, we can represent the sentence

$$(P \vee Q) \wedge (\neg P \vee R)$$

in Haskell as

[["P","Q"], ["-P","R"]]

Since every sentence of propositional logic is logically equivalent to a conjunction of clauses, the CNF representation in Haskell should be adequate for the needs of this assignment.

Models represent assignments to symbols. A model in the Haskell implementations for this assignment is represented as a list of tuples of Strings and Booleans. For example, in a domain with symbols A, B and C, one model of the world where A is assigned True, B is assigned False and C is True is represented as

[("A",True),("B",False,),("C",True)]

The following type synonyms have been defined in the Inf2d.hs Haskell source file :

- **Symbol**
- **Clause**
- **Sentence**
- **Model**

5 Task 1: Convert to CNF (2 marks)

You need to convert the following propositional logic statement to CNF and express in Haskell using the representation discussed in section 4 above.

$$B_{1,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

Assign this sentence to the variable :

wumpusFact :: Sentence

You should represent the symbols $Z_{x,y}$ as *"Zxy"* in Haskell. For example, the symbol $B_{1,1}$ is expressed as *"B11"* in Haskell.

6 Task 2: General Helper Functions (10 Marks)

In this part of the assignment, you are asked to create helper functions that will be used in several other methods.

- **lookupAssignment :: Symbol → Model → Maybe Bool**

  This function finds the assigned literal to a symbol from given model. It returns *Nothing* if the symbol is not found in the model or returns *Just True* or *Just False* depending on the True/False literal assigned to the symbol in the model.

- **negateSymbol :: Symbol → Symbol**

  This function takes a symbol as input and negates it. For example, if "-P" is given as input, it should return "P".

- **isNegated :: Symbol → Bool**

  For a given symbol, this function checks if it is negated.

- **getUnsignedSymbol :: Symbol → Symbol**

  This function takes a symbol and returns an Symbol without any negation sign if the original symbol had one.

- **getSymbols :: [Sentence] → [Symbol]**

  This function returns a list of all symbols (without duplicates) in all input sentences.

7 Task 3: Truth-Table Enumeration and Entailment (40 Marks)

In this part of the assignment, you are asked to implement the truth-table enumeration algorithm (see figure 1) for deciding propositional entailment. The algorithm takes as input a knowledge base (KB), which is represented as a list of sentences, and a query, which is a sentence in propositional logic. The algorithm returns *True* if the query holds in some model, or *False* if it does not hold in any model.

7.1 Model Enumeration (5 Marks)

In this part of the assignment, you are asked to implement a function which enumerates all models for a given list of symbols. Models are enumerated in a depth-first manner for the given list of symbols. Symbols can only be assigned either *True* or *False*. You should complete the following function:

function TT-ENTAILS?(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $symbols \leftarrow$ a list of the proposition symbols in KB and α
  return TT-CHECK-ALL($KB, \alpha, symbols, \{\}$)

---

function TT-CHECK-ALL($KB, \alpha, symbols, model$) **returns** *true* or *false*
  if EMPTY?($symbols$) **then**
    if PL-TRUE?($KB, model$) **then return** PL-TRUE?($\alpha, model$)
    else return *true // when KB is false, always return true*
  else do
    $P \leftarrow$ FIRST($symbols$)
    $rest \leftarrow$ REST($symbols$)
    return (TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = true\}$)
      and
      TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = false\}$))

Figure 1: Truth-table enumeration algorithm for deciding propositional entailment.

- **generateModels :: [Symbol] → [Model]**

  It takes as input a list of symbols, and returns a list of models (all possible assignments of *True* or *False* to the symbols.)

7.2 Propositional Sentence Evaluation (10 Marks)

This section evaluates propositional sentences when given a model of symbol assignments. You have to complete the functions below:

- **pLogicEvaluate :: Sentence → Model → Bool**

  This function evaluates the truth value of a propositional sentence using the symbols assignments in the model.

- **plTrue :: [Sentence] → Model → Bool**

  This function checks the truth value of list of a propositional sentence using the symbols assignments in the model. It returns true only when all sentences in the list are *true*.

7.3 Propositional Entailment (25 Marks)

- **ttCheckAll :: [Sentence] → Sentence → [Symbols] → [Model]**

  This function takes as input a knowledge base (i.e. a list of propositional sentences), a query (i.e. a propositional sentence), and a list of symbols.

  ttCheckAll recursively enumerates the models of the domain based on the list of symbols to check if the query is satisfied by every model that satisfies the knowledge base. It returns a list of all such models.

- **ttEntails :: [Sentence] → Sentence → Bool**

  In this function, you are asked implement the truth-table enumeration algorithm for deciding propositional entailment as shown in figure 1. A knowledge base and a query are inputs for this function. This function uses **ttCheckAll** to determine if models satisfy the query whenever they satisfy the knowledge base.

- **ttEntailsModels :: [Sentence] → Sentence → [Model]**

  This function, similar to **ttEntails** implements the truth-table enumeration algorithm for deciding propositional entailment as shown in figure 1. It also takes a knowledge base and a query as inputs. However, it returns a list of all models for which the knowledge base entails the query.

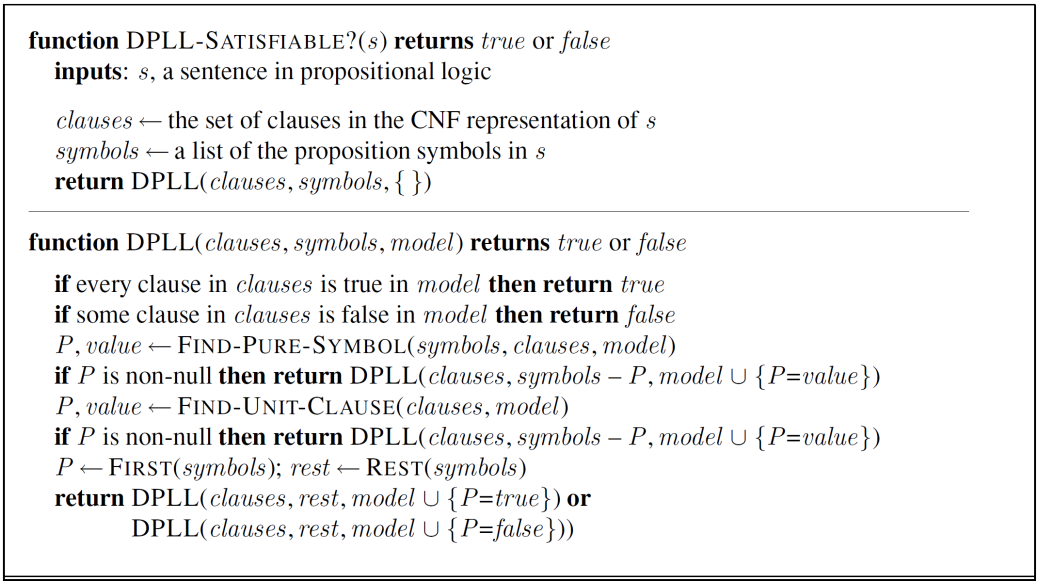

Figure 2: DPLL Algorithm for checking satisfiability of a sentence in propositional logic

8 Task 4: DPLL (43 Marks)

The DPLL (DavisPutnamLogemannLoveland) algorithm is a more efficient algorithm than truth-table entailment for checking satisfiability (i.e. entailment). It uses clauses in CNF representation as described in section 4. In this section, you will implement the DPLL algorithm (figure 2). You should refer to R&N Chapter 7 Section 6.1 or NIE Chapter 6 (Logical Agents) Section 6.1.

8.1 Early Termination (8 Marks)

- **earlyTerminate :: Sentence → Model → Bool**

  The early termination function checks if a sentence is true or false even with a partially completed model. For example, given a sentence [[“P”, “Q”], [“P”, ‘R”]] and the molel [(“P”, $True$)], the earlyTerminate function should return *true* since the partially completed model (without assignments for Q and R) still results in the sentence evaluating to *true*.

8.2 Pure Symbol Heuristic (10 Marks)

- **findPureSymbol :: [Symbol] → [Clause] → Model → Maybe (Symbol, Bool)**

  A pure symbol is a symbol that always appears with the same ”sign” in all clauses. This function can also ignore clauses which are already known to be true in the model. The function takes a list of symbols, a list of clauses and a model as inputs. It returns *Just* a tuple of a symbol and the truth value to assign to that symbol. If no pure symbol is found, it should return *Nothing*.

8.3 Unit Clause Heuristic (10 Marks)

- **findUnitClause :: [Clause] → Model → Maybe (Symbol, Bool)**

  A unit clause is a clause with just one literal. It is also a clause in which all literals but one are already assigned false by the model. The **findUnitClause** function takes a list of clauses and a model as inputs. It returns *Just* a tuple of a symbol and the truth value to assign to that symbol. If no unit clause is found, it should return *Nothing*.

8.4 DPLL Satisfiability (15 Marks)

- **dpll :: [Clause] → [Symbol] → Bool**

  The **dpll** function checks the satisfiability of a sentence in propositional logic. It takes as input a list of clauses in CNF and a list of symbols for the domain. It returns *true* if there is a model which satisfies the propositional sentence. Otherwise it returns *false*. Your can use the **earlyTerminate**, **findPureSymbol** and **findUnitClause** functions to complete this function. Also note that DPLL operates over partial models, and so it is important to implement it using a backtracking search algorithm as shown in figure 2.

- **dpllSatisfiable :: [Clause] → Bool**

  This function serves as the entry point to the dpll function. It takes a list clauses in CNF as input and returns true or false. It uses the dpll function above to determine the satisfiability of its input sentence.

9 Task 5: Evaluation (5 Marks)

DPLL is a more efficient algorithm for checking satisfiability of a propositional sentence than the truth table enumeration algorithm. In this part of the assignment, you are asked to evaluate your algorithms to confirm this statement. You have to define two items for evaluation: a knowledge base (i.e. sentences in propositional logic), and a query sentence. Both items should have their clauses in CNF representation and should be assigned to the following variables respectively:

- **evalKB :: [Sentence]**
- **evalQuery :: Sentence**

Find the average run times(in milliseconds) of the **ttEntails** and **dpllSatisfiable** functions. Assign these run time values to the variables:

- **runtimeTtentails :: Double**
- **runtimeDpll :: Double**

Which performs better? Your implementation of DPLL algorithm or the truth table enumeration algorithm for propositional entailment?

10 Notes

Please note the following:

- To ensure maximum credit, make sure you have commented your code and that it can be properly loaded on ghci before submitting.
- All the algorithms are expected to terminate within 1 minute or less in this assignment. If any of your algorithms take more than 1 minute to terminate, you should make a comment in your code with the name of the algorithm, and how long it takes on average. You may lose marks if your algorithms do not terminate in under 2 minutes of runtime for any of the problems.
- You are free to implement any number of custom functions to help you in your implementation. However, you must comment these functions explaining their functionality and why you needed them.
- Do not change the names or type definitions of the functions you are asked to implement. You may only edit and submit your version of the **Inf2d.hs** file.
- You are strongly encouraged to create your own unit tests to test your individual functions (these do not need to be included in your submitted file).
- Ensure your algorithms follow the pseudocode provided in the books and lectures. Implementations with increased complexity may not be awarded maximum credit.

11 Good Scholarly Practice

Please remember the University requirement as regards all assessed work for credit. Details about this can be found at:

http://www.ed.ac.uk/academic-services/students/undergraduate/discipline/academic-misconduct

and at:

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (generally permitting access only to yourself, or your group in the case of group practicals).

12 Submission

You should submit your copy of the file *Inf2d.hs* that contains your implemented functions using the following command:

submit inf2d 1 Inf2d.hs

Your file must be submitted by **4pm** on the **3rd of March 2016**.