Coursework 1

Propositional Model Checking and Satisfiability

Informatics 2D

Kobby K.A. Nuamah k.nuamah@ed.ac.uk 28 January 2016



Aim

 Understand inference and satisfiability problems in Propositional Logic Δ

- Familiarize yourself with some algorithms for solving inference/SAT problems
- Implement satisfiability algorithms using Haskell

Getting started

- Download coursework file : <u>http://www.inf.ed.ac.uk/teaching/courses/inf2d/course</u> <u>work/Inf2dAssignment1.tar.gz</u>
- Extract content using

tar -xvf Inf2dAssignment1.tar.gz

• Your algorithm implementations and auxiliary functions go into the file:

Inf2d.hs

• Remember to type in your matriculation number at the top of file.

Informatics 2D

Coursework Overview

- You will implement two main algorithms for solving satisfiability problems in Propositional Logic:
 - Model Checking (Truth-Table Enumeration and Entailment)
 DPLL

Informatics 2D

- Your algorithms and several of their key functions will be implemented and tested in a Haskell.
- Inf2d.hs file has the type declarations for each algorithm that has to be implemented. These declarations will inform you about the arguments of the function and the expected output. Do not change the type declarations.

Informatics 2D

Objectives & Grading

- This assignment has 5 core tasks, which will be graded as follows
 - Task 1: Convert to CNF (Representation) [2 marks]
 - Task 2: General Helper Functions [10 marks]
 - Task 3: Truth-Table Enumeration and Entailment [40 marks]

Informatics 2D

- Task 4: DPLL [43 marks]
- Task 5: Evaluation [5 marks]

Testing Your Implementation

- Launch GHCi (or WinGHCi) and run the following commands (where > is the prompt in GHCi):
- > :cd <path to your project dir>
- > :load "Main.hs"
- This compiles all dependent modules of the program.
- Run the application by using the command
- > main

Representation

- Logical sentences in this coursework will be represented in Conjunctive Normal Form (CNF).
- A sentence is expressed as a conjunction of clauses.
- A clause is expressed as a disjunction of literals.
- A symbol is represented as a String in Haskell.
- Clauses are represented by a list of symbols.
- Conjunctions are represented as a list of clauses.
- A model (assignments to symbols) is represented as a list of tuples of Strings and Booleans.

Informatics 2D

Representation (Examples)

- Symbol P is expressed as "P" in Haskell.
- ¬*P* is expressed as "-P"
- The clause *P* \lor *Q* is represented by ["P", "Q"]
- A ∨ ¬B ∨ C is represented in Haskell as ["A", "-B", "C"]
- (*P* ∨ *Q*) ∧ (¬*P* ∨ *R*) becomes [["P","Q"],["-P","R"]]
- Consider a domain with symbols A, B and C, and model of the world where A is True, B is False and C is True. This model is expressed as: [("A",True),("B",False),("C",True)]

Informatics 2D

Task 1 : Convert to CNF

• In this task, you will convert the propositional fact (sentence) below from the Wumpus world into CNF and express this in Haskell.

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

• The sentence in CNF should be assigned to the variable:

wumpusFact :: Sentence

• The symbol $B_{1,1}$ should be expressed as "B11" in Haskell.

Informatics 2D

Task 2: General Helper Functions

- Functions defined in this task will be useful in the remaining tasks.
 - lookupAssignment :: Symbol -> Model -> Maybe Bool

Informatics 2D

- negateSymbol :: Symbol -> Symbol
- isNegated :: Symbol -> Bool
- getUnsignedSymbol :: Symbol -> Symbol
- getSymbols :: [Sentence] -> [Symbol]

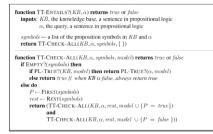
3

9

11

Task 3: Truth-Table Enumeration & Entailment

 Impementation of the truth-table enumeration algorithm for deciding propositional entailment.



Informatics 2D

Task 3 (continued)

Functions to be implemented:

- generateModels :: [Symbol] -> [Model]
- pLogicEvaluate :: Sentence -> Model -> Bool
- plTrue :: [Sentence] -> Model -> Bool
- ttCheckAll :: [Sentence] -> Sentence -> [Symbols] -> [Model]
- ttEntails :: [Sentence] -> Sentence -> Bool
- ttEntailsModels :: [Sentence] -> Sentence -> [Model]

Informatics 2D

12

10

Task 4: DPLL

• DPLL is expected to be more efficient than truth-table enumeration for checking satisfiability. The DPLL algorithm

is shown below.

$clauses \leftarrow$ the set of clauses in the CNF represen $symbols \leftarrow$ a list of the proposition symbols in s return DPLL (clauses, symbols, {})	tation of s
function DPLL(clauses, symbols, model) returns	true or false
if every clause in clauses is true in model them if some clause in clauses is take in model them P, value → FND-PURE-SYMBOL(symbols, clause) if P is non-null then return DPLL(clauses, sup P, value → FND-UNT-CLAUSE(clauses, nucle if P is non-null then return DPLL(clauses, sup P → FIRST-BURNOS); ret = REST(symbols) return DPLL(clauses, rest, model ∪ {P=hreits}) DPLL(clauses, rest, model ∪ {P=hreits})	return false scss, model) bols $= P, model \cup \{P=value\})$ l) bols $= P, model \cup \{P=value\})$) or

15

13

Task 5: Evaluation

- This task focuses on checking the efficiency of DPLL over Truth-Table enumeration for checking satisfiability of propositional sentences.
- To do this, you must create your own set of facts (propositional sentences) and assign to the variable

evalKB::[Sentence]

 You will also define a query to be checked for entailment based on your facts. This sentence should be assigned to the variable

evalQuery::Sentence

All sentences are expected to be in CNF.

Task 4: DPLL (continued)

• The efficiency of the DPLL is as a result of three key heuristics: early termination, pure symbol and unit clause heuristics. (See R&N Ch.7).

14

16

- This tasks implements the DPLL algorithm with the heuristic functions. These are:
 - earlyTerminate :: Sentence -> Model -> Bool
 - findPureSymbol :: [Symbol] -> [Clause] -> Model -> Maybe (Symbol, Bool)
 - findUnitClause :: [Clause] -> Model -> Maybe (Symbol, Bool)

Informatics 2D

- dpll :: [Clause] -> [Symbol] -> Bool
- dpllSatisfiable :: [Clause] -> Bool

Task 5: Evaluation (continued)

- To evaluate your algorithms, run both TT-Entails and DPLL algorithms using the facts in evalKB and the query in evalQuery.
- Do this a number of times and find the average run time in milliseconds for each algorithm.
- Assign the average run time values to

runtimeTtentails::Double
 runtimeDpll::Double

Informatics 2D

17

19

Notes:

- Read the assignment sheet carefully before starting.
- Code clarity is important. Comment you code adequately.
- Reuse functions as much as possible.
- You can add your own helper functions, but you should explain why they are necessary in your comments.
- Test code and make sure they run before submitting.
- Deadline:

3rd March 2016

@ 4pm

Informatics 2D

Haskell

- Purely functional! : "Everything is a function"
- Main topics:
 - Recursion
 - Currying
 - Higher-order functions
 - List processing functions such as map, filter, foldl, sortBy, etc

Informatics 2D

· The Maybe monad

More on Haskell:

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

Haskell Refresher

Informatics 2D

Kob	by. K.A. Nuamah	
	the UNIVERSITY JEDINBURGH	
	Informatics 2D	

Types

- Unlike other programming languages like Java, Haskell has type inference.
- However, type declarations ensures that you are specific about the input arguments of your function and the output values.
- Example:

pLogicEvaluate :: Sentence -> Model -> Bool

 The pLogicEvaluate function takes arguments of type Sentence and Model and returns a Bool type.

Informatics 2D

20

18

22

24

Type Synonyms

type Symbol = String

type Model = [(Symbol, Bool)]

• The type Symbol is a synonym for a String, and type Model is a synonym for a list of (Symbol, Bool) tuples.

Informatics 2D

• Types synonyms are good for code clarity.

Recursion

- Important role in Haskell.
- A function is recursive when one part of its definition includes the function itself again.
- It is important to have a termination condition to avoid infinite loop.

length :: [a] -> Int
length [] = 0
length (x:xs) = 1 + length xs

Informatics 2D

21

Currying

- The process of creating intermediate functions when feeding arguments into a complex function.
- Note: all functions in Haskell really only take one argument
- Example:
 - 2 * 3 in Haskell:
 - (*) function takes first argument 2, and returns an intermediate function (2*)
 - The new function (2*) takes one argument,3, and completes the multiplication
- Applying only one parameter to a function that takes two parameters returns a function that takes one parameter

Informatics 2D

Higher-Order Functions

- Functions are just like any other value in Haskell.
- Functions can take functions as parameters and also return functions.

map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x: xs) = f x : map f xs

• Map takes a function and list and applies that function to every element in the list.

Informatics 2D

26

List Processing Functions (map, filter, foldl, etc.)

• **map** : takes a function and list and applies that function to every element in the list.

map :: (a -> b) -> [a] -> [b]

25

27

• **filter**: takes a predicate (function that returns true or false) and a list and then returns the list of all elements that satisfy the predicate.

filter:: (a -> Bool) -> [a] -> [a]

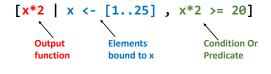
• **fold1**: takes a binary function, an accumulator and a list. It 'folds' up the items in the list and return a single value.

foldl:: $(a \rightarrow b \rightarrow a) \rightarrow a \rightarrow [b] \rightarrow a$

Informatics 2D

List Comprehension

- Build more specific sets out of general sets.
- Example: to create a list of integers that are multiples of 2 and greater than than 20:



Informatics 2D

Maybe Monad

- The Maybe monad represents computations which might "go wrong" by not returning a value.
- If a value is returned, it uses **Just a**, where **a** is the type of the value.
- If no value is available, it returns Nothing.
- Example:

Informatics 2D



3rd March 2016 @ 4pm

Good Luck!!!

Informatics 2D