02157 Functional programming

Michael R. Hansen
DTU Compute
November 15, 2016

Mandatory assignment 4: Solver for Propositional Logic

This is the fourth of four mandatory assignments in 02157 Functional programming. In this assignment you should make a simple solver for Propositional Logic. The purpose of the exercise is to cover basic program construction principles involving finite trees and sets.

It is a requirement for exam participation that 3 of the 4 mandatory assignments are approved. The mandatory assignments can be solved individually or in groups of 2 or 3 students. If the students in a group do not contribute equally, the role of each student must be explicitly stated in the report.

- Your solution should be handed in **no later than Thursday, November 24, 2016**. Submissions handed in after the deadline will face an *administrative rejection.*
- To submit you should upload two files to CampusNet under Assignment 4.:
  1. An F# file (*file*.`fsx` or *file*.`fs`). This file should start with full names and study numbers for all members of the group. Your solution should be submitted as a complete program that can be uploaded to F# Interactive without encountering compilation errors. Failure to comply with that will result in an *administrative rejection of your submission.* Your solution should contain suitable tests of your functions.
  2. A PDF-file (*file*.`pdf`) containing just **one** page with the names and study numbers for all members of the group. This page must also contain a formulation of the Knight-Knave puzzle in propositional logic (Question 7) together with a solution.
  3. Be careful that you submit the right files. A submission of a wrong file will result in an *administrative rejection of your submission.*
- Do not use imperative features, like assignments, arrays and so on. Failure to comply with that will result in an *administrative rejection of your submission.*
- **DO NOT COPY solutions** from others and **DO NOT SHARE your solution** with others. Both cases are considered as fraud and will be reported.

Propositional Logic

In this assignment you shall consider formulas of propositional logic, also called *propositions*, which are generated from a set of *atoms* $p, q, r, \dots$ by the use of the well-known operators: *negation* $\neg$, *disjunction* $\vee$ and *conjunction* $\wedge$. An atom can be either true or false, and the meaning of the operators is:

- $\neg a$ is true if and only if a is false,
- $a \vee b$ is true if and only if a is true or b is true or both are true, and
- $a \wedge b$ is true if and only if both a and b are true,

where a and b can be arbitrary propositions.

Any other propositional operator can be expressed using $\neg$, $\vee$, and $\wedge$. (Actually negation together with just disjunction or just conjunction would suffice.) Implication and biimplication (or equivalence), for example, are definable as follows

$$\begin{array}{lll} a \Rightarrow b & \text{is expressed as} & (\neg a) \vee b \\ a \Leftrightarrow b & \text{is expressed as} & (a \Rightarrow b) \wedge (b \Rightarrow a) \end{array}$$

1. Define a type `Prop` for propositions so that the following are values of type `Prop`:
   - `A "p"` representing the atom p,
   - `Dis(A "p", A "q")` representing the proposition $p \vee q$.
   - `Con(A "p", A "q")` representing the proposition $p \wedge q$.
   - `Neg(A "p")` representing the proposition $\neg p$.

2. A proposition is in *negation normal form* if the negation operator just appears as applied directly to atoms. Write an F# function transforming a proposition into an equivalent proposition in negation normal form, using the de Morgan laws:

   $$\begin{array}{lll} \neg(a \wedge b) & \text{is equivalent to} & (\neg a) \vee (\neg b) \\ \neg(a \vee b) & \text{is equivalent to} & (\neg a) \wedge (\neg b) \end{array}$$

   and the law: $\neg(\neg a)$ is equivalent to a.

3. A *literal* is an atom or the negation of an atom and a *basic conjunct* is a conjunction of literals. A proposition is in *disjunctive normal form* if it is a disjunction of basic conjuncts. Write an F# function which transforms a proposition in negation normal form into an equivalent proposition in disjunctive normal form using the distributive laws:

   $$\begin{array}{lll} a \wedge (b \vee c) & \text{is equivalent to} & (a \wedge b) \vee (a \wedge c) \\ (a \vee b) \wedge c & \text{is equivalent to} & (a \wedge c) \vee (b \wedge c) \end{array}$$

4. We shall use a set-based representation of formulas in disjunctive normal form. Since conjunction is commutative, associative and $(a \wedge a)$ is equivalent to a it is convenient to represent a basic conjunct bc by its set of literals `litOf`(bc).

   Similarly, we represent a disjunctive normal form formula a:

   $$bc_1 \vee \ldots \vee bc_n$$

   by the set

   $$\texttt{dnfToSet}(a) = \{\texttt{litOf}(bc_1), \ldots, \texttt{litOf}(bc_n)\}$$

   that we will call the *dns set* of a.

   Write F# declarations for the functions `litOf` and `dnfToSet`.

5. A set of literals ls (and the corresponding basic conjunct) is said to be *consistent* if it does not contain both literals p and $\neg p$ for any atom p. Otherwise, ls (and the corresponding basic conjunct) is said to be *inconsistent.* An inconsistent basic conjunct yields `false` regardless of the truth values assigned to atoms. Removing it from a disjunctive normal form formula will therefore not change the meaning of that formula.

   Declare an F# function `isConsistent` that checks the consistency of a set of literals. Declare an F# function `removeInconsistent` that removes inconsistent literal sets from a dns set.

6. A proposition is *satisfiable* if it is true for some *assignment* of truth values to the atoms.

   A formula in disjunctive normal is satisfiable when one (or more) of its basic conjunctions are. Therefore, the set of satisfying assignments of a proposition can be derived from the consistent literal sets of its disjunctive normal form.

   Declare an F# function `toDNFsets` that transforms an arbitrary proposition into a dns set with just consistent literal sets.

   Declare a function `impl` a b representing the implication $a \Rightarrow b$ and a function `iff` a b representing the biimplication (equivalence) $a \Leftrightarrow b$.

   Use `toDNFsets` to determine the satisfying assignments for the following two formulas:

   - $((\neg p) \Rightarrow (\neg q)) \Rightarrow (p \Rightarrow q)$
   - $((\neg p) \Rightarrow (\neg q)) \Rightarrow (q \Rightarrow p)$

   where p and q are atoms.

7. In this question you shall solve a *Knights and Knaves* puzzle, which is a kind of puzzle originating from the logician R. Smullyan. The general theme addresses an island that is inhibited by two kinds of citizens: Knights, who always tell the truth, and knaves, who always tell lies. On the basis of utterances from some inhabitants you must decide what kind they are.

   You may find many Knight and Knave puzzles on the internet. The following originates from `http://www.homeschoolmath.net/reviews/eimacs-logic.php`.

   Three inhabitants: a, b and c, are talking about themselves:

   - a says: "All of us are knaves."
   - b says: "Exactly one of us is a knight."

   To solve the puzzle you should determine: What kinds of citizens are a, b and c?

   Solve this puzzle by modelling the two utterances above in propositional logic and using `toDNSsets` to find the satisfying assignments. (Hint: Use a to describe that a is a knight and $\neg a$ to describe that a is a Knave.)

8. The satisfiability problem for propositional logic is an NP-complete problem, and the above transformation to disjunctive normal form propositions or to dns sets has in the worst case an exponential running time.

   The disjunctive normal form of the following proposition:

$$(p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \cdots \wedge (p_n \vee q_n) \tag{1}$$

   will, for example, have 2^n basic conjuncts.

   Declare an F# function `badProp` n representing proposition (1).

   Compute `toDNFsets(badProp` n`)` for a small number of cases and check that the resulting sets indeed have 2^n elements.