CS:4420 Artificial Intelligence
Spring 2019

Homework 3

Part A

Due: Friday, Mar 15 by 11:59pm

This assignment has two parts, A and B, both to be done *individually*. This document describes Part A which consist of OCaml programming problems.

Download the accompanying OCaml source file `hw3A.ml` and enter your solutions in it where indicated. When you are done, submit it on ICON with the same name. Make sure you *write your name in that file* where indicated.

Each of your answers *must be free of static errors*. You may receive no credit for problems whose code contains syntax or type errors.

Pay close attention to the specification of each problem and the restrictions imposed on its solution. Solutions ignoring the restrictions may receive only partial credit or no credit at all.

You are welcome to use as needed your own helper methods or those provided in `hw3A.ml`. You are allowed in this problem to use any OCaml library methods you want. Those for lists may be most helpful.[1] Use recursion, pattern matching and combinators to write clean code.

1 Propositional Logic in OCaml

In this problem we model the language of propositional logic in OCaml. In particular, we will use the following types:

```
type id = int

type interpr = id list

type prop = True
          | False
          | P of id
          | Not of prop
          | Or of prop * prop
          | And of prop * prop
          | Impl of prop * prop
          | Iff of prop * prop
```

[1] `https://msdn.microsoft.com/library/a2264ba3-2d45-40dd-9040-4f7aa2ad9788`

```
type answer = Yes | No | Unknown
```

The union type `prop` models propositional sentences (or, more concisely, *propositions*). Propositional symbols (variables) are encoded as terms of the form `(P n)` where n is a positive integer identifier (id). More complex propositions are constructed using the constants `True` and `False`, the unary constructor `Not`, and the binary constructors `Or`, `And`, `Impl`, and `Iff` standing respectively for the propositional connectives $\vee$, $\oplus$, $\wedge$, $\Rightarrow$, and $\Leftrightarrow$. The type `interpr` implements propositional interpretations as a OCaml list of propositional symbol ids (with no repetitions). A propositional symbol `(P n)` is meant to be true in an interpretation i if and only if n occurs in i.

Using these types, implement the methods below.

1. Write a function `meaning: interpr -> prop -> bool` that takes an interpretation i and a proposition p, and returns the OCaml Boolean value `true` if the interpretation satisfies the proposition according to the semantics of propositional logic, and returns the value `false` otherwise.

   For example, `meaning [1; 2] (And (P 1, P 2))` is `true`, while
   `meaning [1] (And (P 1, P 2))` is `false`.

2. The code in `hw3A.ml` defines a method `isValid: prop -> answer` that takes a proposition `p`, and returns `Yes` if `p` is valid and `No` otherwise. To do that, `isValid` generates all possibles interpretations of `p`'s symbols and checks that each of those interpretations satisfies `p`. It uses both the function `meaning` above and the function `vars: prop -> id list` that takes a proposition `p` and returns a list, with no repetitions, of all the ids of the propositional symbols in `p`. Implement `vars`.

   For example, `vars (And (P 1, Not (P 2)))` is `[1; 2]` (`[2; 1]` is also acceptable as the order of the ids does not matter).

3. Write a function `isUnsat: prop -> answer` that takes a proposition `p`, and returns `Yes` if `p` is unsatisfiable in propositional logic and returns `No` otherwise.

4. Write a function `entails: prop list -> prop -> answer` that takes a list `ps` of propositions and a proposition `p`, and returns `Yes` if the set of propositions in `ps` entails `p` in propositional logic, and returns `No` otherwise.

5. Write a function `areEquiv: prop -> prop -> answer` that takes two propositions `p1` and `p2`, and returns `Yes` if `p1` and `p2` are equivalent in propositional logic and returns `No` otherwise.

2 Rule Soundness

If you were unsure about the soundness of the inference rules described in Chap. 7.5 of the textbook and in the related course notes, you could use some of the code implemented in the previous section to verify their soundness mechanically. How would you do that?

1. Explain that in an OCaml comment.