1. (Semantics of Propositional Formulas)

Here is abstract syntax for truth-tables ($t$) and propositional formulas ($f$):

\[
\begin{align*}
    t & ::= \cdot \mid t, x = c \\
    f & ::= c \mid x \mid \neg f \mid f \land f \mid f \lor f \mid f \to f \mid f \iff f \\
    c & ::= \text{true} \mid \text{false}
\end{align*}
\]

Here is an informal semantics: A truth-table maps variables to constants. The “empty” table maps every variable to false. Given a truth-table, a formula evaluates to true or false. A constant formula evaluates to itself. The formula $x$ is evaluated using the truth-table in the obvious way. The logical connectives are: $\neg$ for not (true when subformula is false), $\land$ for and (true when both subformulas are true), $\lor$ for or (true when either subformula is true), $\to$ for implies (true if the left subformula is false or the right subformula is true), and $\iff$ (true if the subformulas evaluate to the same result).

The code provided to you implements the following concrete syntax: A file contains one truth-table followed by one formula. The truth-table is surrounded in parentheses and entries are not separated by commas. Examples include $(x = #t \ y = #f)$ and $()$, where the latter is the empty table. Uses of binary operators must be surrounded by parentheses whereas constants, variables, and negation must not be surrounded by parentheses. Constants are written #t (true) or #f false. Operators are written NOT, AND, OR, -->, or <-- (for $\neg$, $\land$, $\lor$, $\to$, $\iff$ respectively). Variables are one or more lower-case letters. For example, the table $\cdot$, $x = \text{true}$, $y = \text{false}$ and formula $\neg((x \iff (y \land \text{true})) \lor (y \to \text{false})$ are written as:

$$(x = \#t \ y = \#f)$$
$$\text{NOT} ((x \iff (y \text{ AND } #t)) \text{ OR } (y \to \#f))$$

(a) Give a large-step operational semantics for propositional formulas.

(b) Implement a “large-step operational semantics” interpreter in O’Caml by changing the definition of `Main.interpret`. Your function should return `True` or `False` (the abstract syntax for the constants).

(c) Implement a “denotational” translation from propositional formulas and truth tables into O’Caml by changing the definitions of `Main.denote_formula` and `Main.denote_table`. The return type of `Main.denote_table` should be `string -> bool` (i.e., a function from O’Caml string to O’Caml booleans). The return type of `Main.denote_formula` should be `(string->bool)->bool`. The result of your denotation should not refer to any abstract syntax (except variables are still strings).

Note: Change only `main.ml` and observe the comment “(*Do not change anything below here*)”

2. (Nondeterministic IMP)

We add a new statement form to IMP, written $s_1 \parallel s_2$. We provide two different semantics.

Here is Extension $A$:

\[
\begin{align*}
    A1 & \quad H; s_1 \to H'; s'_1 \\
    A2 & \quad H; s_2 \to H'; s'_2 \\
    A3 & \quad H; s_1 \parallel s_2 \to H'; s'_1 \parallel s'_2
\end{align*}
\]

Here is Extension $B$:

\[
\begin{align*}
    B1 & \quad H; s_1 \to H'; s'_1 \\
    B2 & \quad H; s_1 \parallel s_2 \to H; s_1 \\
    B3 & \quad H; s_2 \to H; s_2
\end{align*}
\]
(a) Prove that Extension A makes IMP nondeterministic. Give full derivations. (That is, prove there exists a program $s$ where $s$ can produce a heap where $ans$ has two different values or there exists an $s$ that can terminate or not terminate.)

(b) Prove that Extension B makes IMP nondeterministic. You may give full derivations or just a sequence of $H; s$ pairs.

c) Prove or disprove: For all $H$ and $s$, if $H; s$ might diverge under Extension A (i.e., there exists an infinite sequence $H; s \rightarrow H_2; s_2; \rightarrow H_3; s_3; \ldots$), then $H; s$ might diverge under Extension B.

d) Prove or disprove: For all $H$ and $s$, if $H; s \rightarrow^* H'; \text{skip}$ under Extension A, then $H; s \rightarrow^* H'; \text{skip}$ under extension B. (I.e., there exists an execution sequence producing the same heap. We are not asking if extension B must always produce $H'$.)

e) Prove or disprove: For all $H$ and $s$, if $H; s$ might diverge under Extension B, then $H; s$ might diverge under Extension A.

f) Prove or disprove: For all $H$ and $s$, if $H; s \rightarrow^* H'; \text{skip}$ under Extension B, then $H; s \rightarrow^* H'; \text{skip}$ under Extension A.

Hints:

- You can use the same key lemma for parts (c) and (d). These problems are not easy.
- Part (f) is by far the most difficult. Think of it as extra credit. Attempt it only after completing the rest of the assignment. Use this auxiliary judgment and the following two lemmas (which you should prove).

\[
\begin{array}{ccc}
R_1 & R_2 & R_3 \\
R(s, s) & R(s_1; s_2) & R(s_1, s_3) \\
& R(s_1; s_2; s) & R(s_2, s_4) \\
& R(s_1; s, s_2; s) & R(s_1 || s_2, s_3 || s_4) \\
R_4 & R_5 \\
R(s_1, s_4) & R(s_2, s_3) & R(s_1, s_2) \\
& R(s_1 || s_2, s_3 || s_4) & R(s_1, s_2) \\
& R(s_1 || s_2 || s_3 || s_4) & R(s_1, s_2)
\end{array}
\]

Lemma 1: If $R(s, \text{skip})$, then $H; s \rightarrow^* H; \text{skip}$ under Extension A.

Lemma 2: If $R(s_1, s_2)$ and $H; s_2 \rightarrow H'; s_2'$ under Extension B, then there exists an $s_1'$ such that $H; s_1 \rightarrow^* H'; s_1'$ under Extension A and $R(s_1', s_2')$. 