CSE 505, Fall 2003, Assignment 2
Due: 28 October 2003, 10:30AM (firm)

1. (Semantics of Propositional Formulas)
Here is *abstract syntax* for truth-tables (t) and propositional formulas (f):

$$\begin{array}{rcl} t & ::= & \cdot \mid t, x = c \\ f & ::= & c \mid x \mid \neg f \mid f \wedge f \mid f \vee f \mid f \rightarrow f \mid f \leftrightarrow f \\ c & ::= & \texttt{true} \mid \texttt{false} \end{array}$$

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), $\wedge$ for and (true when both subformulas are true), $\vee$ for or (true when either subformula is true), $\rightarrow$ for implies (true if the left subformula is false or the right subformula is true), and $\leftrightarrow$ (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 $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ respectively). Variables are one or more lower-case letters. For example, the table $\cdot, x = \texttt{true}, y = \texttt{false}$ and formula $\neg((x \leftrightarrow (y \wedge \texttt{true})) \vee (y \rightarrow \texttt{false})$ are written as:

```
(x = #t y=#f)
NOT ((x <--> (y AND #t)) OR (y --> #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*:

$$\text{A1}\quad \frac{H\ ;\ s_1 \ \rightarrow\ H'\ ;\ s_1'}{H\ ;\ s_1 \parallel s_2 \ \rightarrow\ H'\ ;\ s_1' \parallel s_2} \qquad \text{A2}\quad \frac{H\ ;\ s_2 \ \rightarrow\ H'\ ;\ s_2'}{H\ ;\ s_1 \parallel s_2 \ \rightarrow\ H'\ ;\ s_1 \parallel s_2'} \qquad \text{A3}\quad \frac{}{H\ ;\ \mathsf{skip} \parallel \mathsf{skip} \ \rightarrow\ H\ ;\ \mathsf{skip}}$$

Here is *Extension B*:

$$\text{B1}\quad \frac{H\ ;\ s_1 \ \rightarrow\ H'\ ;\ s_1'}{H\ ;\ s_1 \parallel s_2 \ \rightarrow\ H'\ ;\ s_1' \parallel s_2} \qquad \text{B2}\quad \frac{}{H\ ;\ s_1 \parallel s_2 \ \rightarrow\ H\ ;\ s_2 \parallel s_1} \qquad \text{B3}\quad \frac{}{H\ ;\ \mathsf{skip} \parallel s_2 \ \rightarrow\ H\ ;\ s_2}$$

(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 $H; s \to H_2; s_2; \to H_3; s_3; \ldots$), then $H; s$ might diverge under Extension B.

(d) Prove or disprove: For all H and s, if $H; s \to^* H'; \mathsf{skip}$ under Extension A, then $H; s \to^* H'; \mathsf{skip}$ 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 $H \ ; \ s \ \to^* \ H' \ ; \ \mathsf{skip}$ under Extension B, then $H \ ; \ s \ \to^* \ H' \ ; \ \mathsf{skip}$ 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).

$$\text{R1}\ \frac{}{R(s, s)} \qquad \text{R2}\ \frac{R(s_1, s_2)}{R(s_1; s,\ s_2; s)} \qquad \text{R3}\ \frac{R(s_1,\ s_3) \qquad R(s_2,\ s_4)}{R(s_1 \parallel s_2,\ s_3 \parallel s_4)}$$

$$\text{R4}\ \frac{R(s_1,\ s_4) \qquad R(s_2,\ s_3)}{R(s_1 \parallel s_2,\ s_3 \parallel s_4)} \qquad \text{R5}\ \frac{R(s_1, s_2)}{R(\mathsf{skip} \parallel s_1,\ s_2)}$$

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

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