CS109A Assignment #6
Due 3:15 PM, 3/6/96

Please remember to separate the written homework from the programming problems and hand in each separately.

Written Homework

H1. (5 points) In the probability space of the deal of two cards, in order, from a 52-card deck, what is the probability that the cards are the same either in rank or in suit? (Exercise 4.9.2(e), FCS)

H2. (10 points) In the 6-spot game of Keno (see p. 190 ff., FCS), what is the probability that a player guesses exactly 4 out of 6 correctly?

H3. (5 points) Using the space of Fig. 4.14 (p. 190, FCS), give the conditional probability that the first die is even, given that the sum of the dice is at least 10.

H4. (10 points) Divide the Chuck-a-Luck (see Example 4.16, p. 182, FCS) probability space into three regions as in Example 4.33 (p. 199). Use this division and Equation (4.10) to compute the probability that there is exactly one 1 showing. (Exercise 4.10.2(c), FCS) *Note*: we shall not cover this formula in class; please read the relevant material, as you should with all assigned sections, even if they are not covered during the lectures or sections.

H5. (10 points) Let E and F be two independent events, and let $\bar{E}$ and $\bar{F}$ be their *complement* events, i.e., the events that E (respectively F) does not occur. Prove that $\bar{E}$ is independent of $\bar{F}$. *Hint*: Consider a diagram such as Fig. 4.15 on p. 193, FCS or the diagram on the first page of the notes for 2/10.

H6. (10 points) Prove by structural induction that in any tree T, the number of leaves is 1 more than the number of nodes that have right siblings. (Exercise 5.5.4, FCS) *Important*: In order to be considered a structural induction

   a) You must not mention numerical properties of the tree such as the height of nodes or the number of nodes in subtrees.

   b) You must use the inductive hypothesis to say something about the children of a node.

Programming Problems

For this assignment, you *must* make a copy of your code in the dropoff directory. You should call your file (or directory if you want to hand in several files) by your first initial, your last name, and the number 6. For example, John Robert Smith would copy his code to the file `/usr/class/cs109a/dropoff/jsmith6`. Remember you can't modify or retrieve what you copy, and if you feel the need to change it at the last minute, copy another file with another name, and tell us what you have done when you hand in the hardcopy.

Under any circumstances, you still need to hand in your programs and scripts in hardcopy as usual.

The purpose of this assignment is to get some experience with a nontrivial data type. The datatype in question is a representation of logic expressions involving the operators `And`, `Or`, `Implies`, and `Not`. Operands are of some arbitrary type called `'operand`. In practice, we'll use either `string` as the operand type, representing an expression with variables such as $xy \to z$, or `bool` as the oeprand type, representing an expression with constants `true` and `false`, e.g., `false`→`true`. The definition of the datatype `'operand le` (logical expression with an operand type) is:

```
datatype ('operand) le =
    And of ('operand le) * ('operand le) |
    Or of ('operand le) * ('operand le) |
    Not of ('operand le) |
    Implies of ('operand le) * ('operand le) |
    Op of 'operand;
```

You may type this definition or download it from the class web page.

Here is an example value of type `string le` "assigned" to a variable called `exp`:

```
val exp = And(Op("x"),Or(Not(Op("y")),Implies(Op("x"),Op("y"))));
```

It represents the logical expression $x(\bar{y} + (x \to y))$. Note that the type constructors such as `And` are binary, prefix operators, that look like functions of two arguments. They are not functions, however, but rather they help build values of the type `'operand le`. Also note that it is possible to tell ML to treat the binary type constructors as infix operators by following the datatype definition by declarations such as:

```
infix 7 And;
infix 6 Or;
infixr 5 Implies;
```

These statements declare `And` to be an infix operator with precedence 7 (the same as multiplication and division in SML/NJ); `Or` becomes an infix operator at precedence level 6 (like +), and `Implies` is a right-associative (that's the "r" in `infixr`) infix operator at level 5 (binds less strongly than +). However, *do not use these declarations*; they will require you to put many pairs of parentheses around subexpressions and you will probably create a lot of trouble for yourself.

P1. (10 points) We will need some additional type (not datatype) declarations in this and the next assignment. Write the following type definitions:

a) `varLe`, which is an `le` with the type `'operand` restricted to be `string`. This type is suitable for expressions with variables as operands.

b) `constLe`, the same, but with type `'operand` restricted to be `bool`. This type is suitable for expressions with logical constants as operands.

c) `truthAsgn`, a list of pairs, each consisting of a `string` and a `bool`. This type is suitable for *truth assignments*, which associate a truth-value (true or false) with each of a list of variables. For example, `[("x",true) ("y",false)]` is the truth-assignment that sets x to `true` and y to `false`.

Hand in your code and a script showing ML's response to your type definitions.

P2. (10 points) Write a function `lookup(s,t)` that takes a `string` s and a `truthAsgn` t and returns the value that appears as the second component of a pair in t whose first component is s. For example, `lookup("y",`t`)` would return `false` if t were the `truthAsgn` of problem P1(c). Define and raise a suitable exception if s is not found in t. You may assume s appears in at most one pair on the list t. Show your code and a script with the response of ML to your function definition and its response to calls `lookup("y",`t`)` and `lookup("foo",`t`)`, where t is the particular `truthAsgn` mentioned in P1(c).

P3. (15 points) Write a function `eval(e,t)` that takes a `varLe` e and a `truthAsgn` t and determines the value of e when variables are replaced by `true` or `false` according to the truth assignment t. Show your code and a script with ML's response to your function and its response to the call `eval(exp, `t`)`, where t is the `truthAsgn` of P1(c) and `exp` is the particular expression mentioned in the introduction to the programming problems.

P4. (15 points) Write a function `printExp(e)` that takes a `varLe` e and prints e nicely indented, with the operators in the middle (vertically). For example, given an expression `And(`e_1`,`e_2`)` that must be indented 3 "tabs", first print e_1 indented 4 tabs, then print the `And` on a separate line, indented 3 tabs, and finally, print e_2 indented 4 tabs. For a `Not` operator, print the `Not` and then the single expression following indented one more tab. The printing of the expression `exp` above should be:

```
        x
And
                Not
                        y
        Or
                        x
                Implies
                        y
```

Note that the outermost operator (`And` in this case) should not be indented at all. Show your code and the response of ML to your function and the result of applying it to `exp`. *Hint*: you may wish to use a "helper" function that takes an additional argument: the number of tabs to indent.