Due: Friday July 6th at 9:30am at the beginning of discussion section - homework will not be accepted after 9:40am so be on time! Late homework will receive no credit. Be sure to label the problem you are solving clearly with the problem number and subsection. Typing your homework is not required, but homework should be legible. Illegible solutions will receive no credit. Be sure to put your name on the homework.

Remember, you must do this homework on your own, without any external sources. If you are unsure what is allowed, please contact the instructor.

1. (30 points) Creating Grammars

Write unambiguous grammars for the following languages:

(a) \{a^n b^n | n \geq 0 \text{ and } n \text{ is odd}\}

(b) All valid OCaml lists created using ::. Make sure your construction correctly shows the right association of this operator. Use \(v(\prime a)\) to indicate a value of type \(\prime a\). Examples of valid strings in this language: 1 :: [], [], "hi" :: "there" :: []

(c) All valid OCaml guards for if statements in the abbreviated language containing terminals: (,) , | | , && , < , > , =. Parenthesis must be appropriately matched with every opening left parenthesis followed by a closing right parenthesis. ( and ) have the highest precedence, followed by \&\& and | | and the lowest precedence is given to <, >, = etc. Use \(v(\prime a)\) to indicate a value of type of type \(\prime a\). Note: You don’t need to worry about type compatibility. Assume that this is done separately.

Answer:

(a) \(S \to aTb\)
\(T \to aaTbb|\epsilon\)

(b) \(S \to [] | v(\prime a) :: S\)

(c) \(G \to G < B | G > B | G = B | G >= B | G <= B | B\)
\(B \to B & & P | B | P | P\)
\(P \to v(\prime a)|(G)\)
2. (10 points) Ambiguity Proofs

Prove that the following grammar is **ambiguous**. Note: We did not call our proofs from class “proofs”, but recall that we showed that grammars were ambiguous by using example strings and the definition of ambiguous.

\[
S \rightarrow ST | SU | c \\
T \rightarrow Ub | b \\
U \rightarrow aT | a
\]

**Answer:**

Here are two parse trees for the string “cab”. Since there are two different parse trees for the same string, the grammar is ambiguous.

3. (10 points) OCaml Types

(a) What is the type of the following OCaml function? Explain **why** this type is correct.

(Yes, you can get the answer to the first part by typing this into a computer... but you will need to know how to do this without a computer for the exam, so think about it anyway.)

```ocaml
let rec func (f, l1, l2) = match l1 with
    [] -> []
  | (h1::t1) -> match l2 with
      [] -> [f h1]
    | (h2::t2) -> [f h1; f h2]
```

(b) Make a function which has the following type:

\[\text{func : (} 'a -> 'b \text{) * (} 'c * 'c \rightarrow 'a \text{) * 'c \rightarrow 'b \text{)}\]

You may check your answer using your computer, but make sure you can do it without this aide.

**Answer:**

(a) \[\text{func : (} 'a -> 'b \text{) * 'a list * 'a list \rightarrow 'b list}\]

Let’s let the type of the first list, \(l1\), be \( 'a \text{ list} \). Then we know that function \(f\) has to take
something of type 'a since it is applied to the head of list 1. Function f is also applied to the head of list 2, so l2 must also be of type 'a list. We know nothing about the transformation that f performs, so the result could be of a different type, so \( f : 'a \rightarrow 'b \) and the result of \( \text{func} \) is type 'b list. Putting all of this together, we get the type shown above.

(b) let rec func (f, g, a) = f (g (a, a))
Immediately we can see that the function is going to take a tuple with three elements (as separated by the * in the type) and return a single element. Looking closer, we see that this return value is going to be the same as the return value of the first element in the tuple, which we also see is a function. Let’s call it f. So the return of func is the same as the return of f, so we probably need to apply f to something and return that value to get the appropriate type. In addition, we see that the second element of our tuple is also a function, which means that we need to apply it to some values (it takes a tuple) in order for OCaml’s type inference to reason that it’s a function. This function, which we’ll call g, takes two elements of the same type and these elements are also the same type as the last item in the tuple (which we’ll call a). So we can force g to have the appropriate type by passing in the tuple (a,a). The return of g is the same as the input to f, so we can pass this directly, resulting in the answer shown above which is of the appropriate type.